CAPACITY EXPANSION OF SINGLE COMMODITY AND MULTICOMMODITY FLOW NETWORKS

A Thesis Submitted
in Partial Fulfilment of the Requirements
for the Degree of
MASTER OF TECHNOLOGY

By

D. N. SARWADE

IMEP-1970-M-SAR-CAP

To

Tirtharup KAKA and Saw. AAI
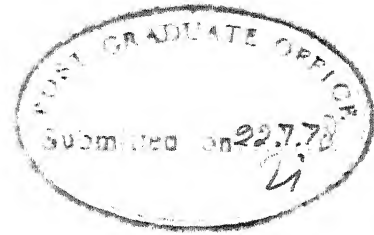
CERTIFICATE

This is to certify that this work on 'CAPACITY EXPANSION OF SINGLE COMMODITY AND MULTICOMMODITY FLOW NETWORKS', by D.N. Sarwade has been carried out under my supervision and has not been submitted elsewhere for the award of any degree.

(A.K. Mittal)
Assistant Professor
Industrial and Management
    Engineering Programme
Indian Institute of Technology
Kanpur 208016

July, 1978.

ACKNOWLEDGEMENTS

CONTENTS
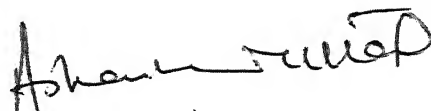
SYNOPSIS

In this thesis problem of capacity expansion in single commodity and multicommodity networks is considered. The problem considered is to increase the capacity of network to accommodate increased flow by increasing the capacity of existing arcs or by introducing new arcs. It is assumed that only cost under consideration is the cost of increasing the capacity of an arc.

For single commodity flow network four heuristics and branch and bound method are developed. Computational performance of these algorithms is tested by randomly generated problems. Computational results of these test problems on IBM 7044/1401 system are presented.

A branch and bound method for multicommodity flow network capacity expansion problem is developed.

CHAPTER I

INTRODUCTION

Physical distribution systems such as electricity or water supply, telephone or microwave links road or rail networks, etc. can be naturally considered as networks of flow. A common feature of these systems is that expansion may be necessary with lapse of time as demand requirements may change. In the case of an electricity power network, let us consider power flow from a specific source node N_s to a sink node N_t and assume that the power requirement of the sink is increased from some value P_1 to another value P_2. The existing network may not be able to accommodate this increased power flow from N_s to N_t and what may be required is to add extra arcs to the network to make such a flow possible. Arcs can be added to the network at a specific cost and capacity, and the problem is then one of choosing which arcs to add so as to increase the maximum flow capacity (from N_s to N_t) to a value at least as large as P_2 and at the same time incur the least cost.

If in a network there is only one source node and one sink node and flow is also of one kind only then this problem is referred to as single commodity flow problem.

If there are many sources and many sinks in the network and if we make the restriction that the flow from certain sources must be sent to certain sinks, then the problem is a multi-commodity flow problem. In case of single commodity flow problem, an undirected arc can always be thought of as two directed arcs of opposite direction, and arc flows of opposite directions can be cancelled. But arc flows of different commodities can not cancel each other. This is one of the central difficulties of the multi-commodity flow problem.

In Chapter II, we present literature survey for capacity expansion problem and other related problems like minimal cost flow problem and fixed charge problem. Chapter III deals with some of the pure network problems, which arise as subproblems in either one of the heuristic methods or the exact method, which are developed in this thesis. Some algorithms available from literature which were used in this work for these problems are also presented. Computer codes for these algorithms were developed and are presented in Appendix B.

In Chapter IV we present mathematical programming formulation and solution methodologies for single commodity flow network capacity expansion problem. The solution methodologies which were developed for this problem

include four heuristic algorithms and an exact method. Large number of problems were solved. Computational results are also presented.

Multi-commodity flow network capacity expansion problem is discussed in Chapter V. Mathematical programming formulation for the problem is presented. An exact method for the solution of the problem is suggested.

CHAPTER II

LITERATURE REVIEW

In this chapter we present review of the available literature for following problems.

1. Shortest path problem

2. Maximal flow problem

3. Minimal cost flow problem

4. Network capacity expansion problem

5. Multi-commodity flow problem, and

6. Fixed charge problem.

These problems are chosen for review of literature because few of them resemble closely with our problem and others arise as subproblems in various methods developed in this work.

2.1 SHORTEST PATH PROBLEM:

Consider a network (V, E), where V is the set of nodes and E the set of arcs. Every arc $A_{ij} \in E$ has associated with it a distance c_{ij}. The problem is to find a path from any node N_k to any other node N_1 with sum of the distances c_{ij} of arcs in the chain a minimum.

More than 400 papers have been published in the area of shortest path problems and allied areas [43] and we have got the least intention to discuss them here. We refer for a detailed bibliography on these to Pierce [43].

Some more recent works published in this area are those of Hadlock [23] and Fredman [15].

Hadlock [23] has developed a shortest path algorithm for grid graphs. Grid graphs are a simple class of planer graphs for which the vertices can be assigned integer coordinates so that neighbours agree in one coordinate and differ by one in the other coordinate. Grid graphs arise in applications from the layout design of integrated circuits to idealized models of city street networks. In many applications, a shortest path between two vertices is needed. The best known algorithms for the shortest path in a general graph of n vertices are of complexity $O(n^2)$. Hadlock's algorithm however taking advantage of the concept of direction present in grid graphs is $O(n)$ in the worst case and $O(\sqrt{n})$ in the best case.

Fredman [15] has shown that $O(n^{5/2})$ comparisons and additions suffice to solve the all pairs shortest path problem for directed graphs on n vertices with non-negative edge weights. This result is exploited to produce an $O(n^3)$ algorithm for solving the shortest path problem.

2.2 MAXIMAL FLOW PROBLEM:

The maximal flow problem can simply be described as finding maximum possible flow from source node N_s to sink node N_t without violating the capacity constraints $x_{ij} \leq b_{ij}$ on arcs.

A labelling algorithm (Ford and Fulkerson [12]) which was used to solve sub-problems in this thesis is described in detail in Section 3.1.

A number of researchers have published their work in this area. Our intention will be to review only recent literature published in this area.

Dinic [11] has given an algorithm which solves the problem exactly in the general case after not more than Cn^2p (machine) operations, where n is the number of nodes of the net, p is the number of arcs in it and c is a constant not depending on the network. For integer data this algorithm, like the algorithm of Ford and Fulkerson gives an integer solution with a supplementary estimate of the number of operations C_1np plus an estimate of the last. This algorithm is an iterative process like Ford and Fulkerson's [12] algorithm.

Karzanov [33] has presented an algorithm with an improved bound $O(n^2)$. Malhotra, Kumar and Maheshwari [38]

has given another algorithm, which is simpler than Karzanov's [12]algorithm to solve the per stage flow problem in $O(n^2)$ steps.

2.3 MINIMAL COST FLOW PROBLEMS:

Consider a connected network consisting of nodes N_i and arcs A_{ij} leading from N_i to N_j. Among the nodes N_i, there is a special node N_s called the source and a special node N_t called the sink. The flow from N_i to N_j in arc A_{ij} is denoted by x_{ij}. Now the minimal cost flow problem can be formulated xas:

$$\min Z = \sum c_{ij} (x_{ij}) \qquad (2.3.1)$$

s.t.

$$\sum_i x_{ij} - \sum_k x_{jk} = \begin{cases} -v & \text{for } j = s \\ 0 & \text{for } j \neq s, t \\ v & \text{for } j = t \end{cases} \qquad (2.3.2)$$

$$x_{ij} \geqslant 0$$

where $c_{ij}(x_{ij})$ are non-negative functions of x_{ij}, and the arc flows x_{ij} are required to be positive integers or zero. The parameter v, which should be a non negative integer, represents the total flow from source to sink. Note that equations (2.3.2) express the conservation of flow at nodes other than the source and the sink. The main computational procedures available for this problem where $c_{ij}(x_{ij})$ are linear functions of x_{ij} are the primal dual

type algorithms of Ford and Fulkerson [14] and Busacker and Gowen [4]. These are dual methods in which feasible flows become available when the computations terminate.

Hu [29] has developed an algorithm where the cost functions $c_{ij}(x_{ij})$ are convex functions of x_{ij}. In spirit this algorithm is closely related to Beale's [2] and Busacker and Gowen's [4] algorithm. Hu's [29] algorithm is discussed in greater detail in Section 3.3 of this dissertation. In the later part of this paper Hu has shown that the problems associated with electrical networks, with increasing capacity of a network under a fixed budget and with Max-flow Min-cut theorem may all be formulated in to minimum cost flow problems in convex-cost networks.

Fulkerson's 'out of kilter' algorithm (described in [14]) is essentially a primal method in that it can be started with a feasible flow or one becomes available at an early stage.

Menon [40] has outlined a primal algorithm for the transhipment problem where transportation costs over each arc are convex function of amount shipped.

Klein [35] has added one more primal method for both minimal cost flow and assignment transportation problems, with slight modification it can also be used for problems

involving convex costs. It shares with other primal
methods, the property that it can be started with a 'good'
solution and a better one is always available in case
early termination of computations is required.

A more recent primal algorithm for minimal cost
flow problems with convex costs is that of Weintraub [53].

2.4 NETWORK CAPACITY EXPANSION PROBLEM:

LeBlanc [37] in his paper has developed a long
range planning model of transportation systems to assist
planners in assessing the impact of various levels of
service in a transportation network. Rail road distribution
networks are studied in particular, and emphasis is placed
on the problem of determining optimal levels of service
(improvements, degradations, abandonements) on each arc
in the existing network. The problem considered is to
optimally design a transportation system; the term design as
used, does not comprehend new networks, but more importantly,
the optimization or 'prunning' of existing transshipment net-
works. The model is formulated so that improvements to the
shipping arcs are included as decision variables. The con-
cave transhipment problem is extended to the case where
coefficients of the shipping cost functions are treated as
decision variables. Thus the model determines optimum
movements of freight between nodes and optimum improvements/

degradations to the network to minimize shipping costs plus maintenance costs on the shipping arcs. The result is a nonconvex, nonconcave transhipment problem which is then converted into a concave minimization problem for which global optimal solutions can be found. Unlike LeBlanc [37], Christofides and Brooker [6] consider the problem of increasing the capacity of existing network where new arcs of finite capacity can be added at certain cost. The problem considered is: Given an existing network, a list of arcs which could be added to the network, the arc costs and capacities, and an available budget, choosing which arcs to add to the network in order to maximize the maximum flow from a source s to a sink t subject to the budgetary constraint.

The paper describes an efficient tree search algorithm using bounds calculated by a dynamic programming procedure which are very effective in limiting the solution space explicitly searched. Computational results for a number of medium sized problems are described and computing times are seen to be reasonable.

Howard and Nemhauser [27] has considered this problem: given a sequence of predicted demands for N time periods, determine the optimal investment decision in each period to minimize a linear investment cost and a strictly convex

cost of capacity. The relationship between capacity and the investment decisions is assumed to be linear, but time varying. Constraints on both the individual decisions and on the sum of decisions are considered. They have also derived an algorithm for solving this problem.

2.5 MULTICOMMODITY FLOW PROBLEMS:

There are two basic approaches which have been employed to develop specialized techniques for multicommodity flow problems, decomposition and partitioning. Decomposition approaches may be further characterized as price directive or resource directive.

A price directive decomposition procedure directs the coordination between a master program and each of several subprograms by changing the objective functions (prices of subprograms). The objective is to obtain a set of prices (dual variables) such that the combined solution for all subproblems yields an optimum for the original problem. There are a number of studies which have proposed variations of this general theme. Few of these are as follows.

Cremeans, Smith and Tyndall [9] give an approach-- an extension of the column generation technique used in the multicommodity flow problem that simultaneously considers network chain selection and resource allocation,

thus making the problem both manageable and optimal. The flow attained is constrained by resource availability and network capacity.

A resource directive decomposition procedure, when applied to a multicommodity network flow problem having K commodities is to distribute the arc capacity among the individual commodities in such a way that solving K subprograms yields an optimal flow for coupled problem. At each iteration an allocation is made and K single commodity flow problems are solved. The sum of the capacities allocated to an arc over all commodities is equal to the arc capacity in the original problem. Hence the combined flow from the solutions of the subproblems provides a feasible flow for the original problem. Optimality is tested and the procedure either terminates or a new arc capacity allocation is developed. The earliest reference of which we are aware that suggests this approach appeared in 1956 (Robacker [47]). However, he did not indicate how this approach could be implemented. Sakarovitch[49], Kennington and Shalaby [34] and Held, Wolf, and Crowder [25] have presented techniques for implementing this approach.

Partitioning approaches are specializations of the simplex method where the current basis is partitioned to

exploit its special structure. These techniques are specializations of the primal, dual, or primal dual simplex method. The papers of Saigal [48], Hartman and Lasdon [24] and Graves and Mcbride [19] deal with primal techniques, while the work of Grigoriadis and White [21] is a dual technique.

Kant [32] presents an algorithm for finding maximal sized sets of flows in a certain class of multicommodity flow networks. The class consists of networks with integer capacity edges and with each node being a source or sink for all but at most one commodity. This new procedure lifts a restriction in Kleitman's [36] algorithm that required flow in edges joining source and sink of the same commodity.

Golden [18] has developed an algorithm for handling nonlinear minimum cost multicommodity flow problems and applied it to a large scale network. The commodities are imports and exports; the cost functions considered are quadratic and convex. The setting considered is a Port Planning Model which seeks to find optimal simultaneous routings through the network while fulfilling requirements both at foreign ports and at domestic hinterlands.

The algorithm involves linearizing the cost function and solving the resulting linear program, which is, in fact a series of shortest route problems.

2.6 Fixed Charge PProblem:

The fixed charge problem was first introduced by Hirsch and Dantzig [26] in 1954. Since then more than forty papers have appeared covering various aspects of the problem. Most of these papers are concerned with the following linear fixed charge problem.

$$\text{Min} \quad Z = \sum_{j=1}^{n} c_j \, x_j + d_j \, y_j$$

$$\text{s.t.} \qquad Ax = b \qquad\qquad\qquad (2.5.1)$$

$$x \geqslant 0$$

where, A is a m x n matrix of real scalars

b is a m vector

x is a n-vector and

$$y_j = \begin{cases} 0 & \text{if} \quad x_j = 0 \\ 1 & \text{if} \quad x_j > 0 \end{cases} \qquad \text{for all } j = 1,2,\ldots, n.$$

The quantities d_j are the fixed charges which are incurred if the corresponding x_j are positive. Eventhough more than forty papers have been published in this area, We still do not have an exact and efficient computational algorithm for linear fixed charge problem.

In the paragraphs below we summarize some important properties of solutions to various fixed charge problems and list various solution techniques used, so far.

The objective function Z of linear fixed charge problem (2.5.1) is concave because $c_j x_j + d_j y_j$ is concave for $x_j \geqslant 0$, and non-negative linear combination of concave functions defined over the same convex set is concave[22]. The constraint set S of (2.5.1) is closed, convex and bounded from below. Hence, due to the fact that the global minima of a concave function over a closed, convex set, bounded from below, is taken at one or more of the extreme points of the set [22]; the optimal solution to (2.5.1) occurs at one or more of the extreme points of S. Unfortunately, for linear fixed charge problem there exists the possibility that a local minima, different from the global minima may occur at another extreme point. When each activity has a positive fixed charge associated with it, every extreme point solution yields a local minima.

In case problem (2.5.1) has a network structure, an extreme point of S corresponds to an acyclic solution, i.e., a solution in which arcs carrying flow donot form a cycle.

Balinski [1] has proved a number of theorems for fixed cost transportation problems and then on the basis of these theorems have presented an approximate procedure for the same.

Various approaches used sofar for solving linear fixed charge problem and its special structures are:

1. Mixed Integer Programming [44]

2. Separable Programming [22]

3. Group Theoretic Approach [45], [46]

4. Extreme point methods [41], [20], [39], [5],[10],[52] [50],[7],[8]

5. Cutting Plane Methods [5], [51]

6. Dynamic Programming [22]

7. Branch and Bound Approach [3], [50]

CHAPTER III

NETWORK ALGORITHMS

In this chapter we will discuss various network algorithms which are used in this dissertation for solving subproblems which arise in various solution methodologies. We will discuss

1. Maximal Flow Problem
2. Shortest Chain Problem, and
3. Minimal Cost Flow Problem.

3.1 MAXIMAL FLOW PROBLEM:

A network can be considered as a pipeline system (Hu [30]) with the arcs representing pipelines, the source being the inlet of the water, the sink being the outlet of water and all other nodes just being junctions between pipelines.

The capacity of each arc can be considered to indicate the cross sectional area of the pipeline with such a pipe line system we are interested in the maximum flow that can be put through it from source to sink.

In order to make above intuitive problem precise, we shall first define a flow in the network. A set of non-negative integers x_{ij} is called a flow in a network if

they satisfy the following constraints:

$$\sum_i x_{ij} - \sum_k x_{jk} = \begin{cases} -v & \text{if } j = s, \\ 0 & \text{if } j \neq s, t \\ v & \text{if } j = t \end{cases} \qquad (3.1.1)$$

$$0 \leqslant x_{ij} \leqslant b_{ij} \qquad \text{(for all } i,j) \qquad\qquad (3.1.2)$$

The v which appears in (3.1.1) is a non-negative number called the value of the flow. Note that (3.1.1) represents the fact that flow is conserved at every node except the source and the sink. Constraint (3.1.2) means that the arc flow x_{ij} is always bounded by the capacity of the arc b_{ij}.

To find the maximum flow value in any network is obviously a linear program with the objective function $v = \sum_j x_{sj}$ and the constraints (3.1.1) and (3.1.2). However as this is a very special case of a linear program, few algorithms are available which are more efficient than the simplex method, which is for general linear programs.

In the following paragraphs a labelling algorithm (Ford and Fulkerson [12]) as described by Hu [30] is presented.

The labelling method starts with any arbitrary flow (zero flow may be used as the starting flow) and then tries to increase the flow value. The algorithm terminates if the flow value can not be increased. The algorithm is a systematic

way of searching all possible flow augmenting paths from N_s to N_t. This is done by giving labels to nodes to indicate the direction that flow in an arc may be increased once a flow augmenting path is found. The flow is increased to its maximum capacity along the path and all the labels on the nodes are erased. We then start to assign new labels to nodes based on the new flow.

The algorithm consists of two steps; Step 1 is the labelling process, which assigns labels to nodes, and Step 2 is flow change. Steps 1 and 2 are iterated until the increase of flow becomes impossible.

STEP 1:

Labelling Process. Every node is always in one of the three states, labelled and scanned, labelled and unscanned, or unlabelled. A node is labelled and scanned if it has a label and we have inspected all its neighbouring nodes (Two nodes are neighbors if both are connected by one arc.) A node is labelled and unscanned if it has a label but not all its neighbors have been inspected. A node is unlabelled if it has no label.

Initially all nodes are unlabelled. A label for node N_j always has two parts. The first part is the index of node N_i, which indicates that we can send flow from N_i to N_j, and the second part is a number which indicates

the maximum amount of flow we can send from source to N_j without violating the capacity restrictions. We first assign the label $[s^+, \in(s) = \infty]$ to N_s. The first label simply says that we can send flow from N_s to itself; the number indicates that there is no upper bound on how much can be sent. The node N_s is now labelled and unscanned and all other nodes are unlabelled. In general, select a node N_j which is labelled and unscanned. Assume N_j has a label of the form $[i^+, \in(j)]$ or $[i^-, \in(j)]$. To all neighbouring nodes N_k of N_j which are unlabelled and for which $0 \leq x_{jk} \leq b_{jk}$ assign the label $[j^+, \in(k)]$ to N_k, where

$$\in(k) = \min[\in(j), b_{jk} - x_{jk}] \qquad (3.1.3)$$

The + and - signs in the first label indicate how the flow should be changed in Step 2. Now all the neighbouring nodes of N_j have labels; N_j is considered to be labelled and scanned and may be disregarded during the rest of this step. (If one inspects all the neighbors of N_j and can not label the neighbours, then also N_j is considered to be a labelled and scanned node). All the nodes N_k are now labelled and unscanned.

Continue to assign labels to neighbours of labelled and unscanned nodes until either N_t is labelled or no more labels can be assigned and N_t is unlabelled. If N_t cannot be

labelled, no flow augmenting path exists and, hence, the flow is maximal. If N_t is labelled a flow augmenting path can be found using Step 2.

STEP 2:

Flow Change: Assume that N_t is labelled $[K^+, \in(t)]$. Replace x_{kt} by $x_{kt} + \in(t)$ and turn to N_k. If N_k is labelled $[j^+, \in(k)]$, replace x_{jk} by $x_{jk} + \in(t)$ and turn to N_j. If N_k is labelled $[j^-, \in(k)]$ replace x_{kj} by $x_{kj} - \in(t)$ and turn to N_j. Continue until N_s is reached. Erase the labels on all the nodes and go back to Step 1. Few other algorithms are also available for the same problem, such as Dinic [11], Karzanov [33] and Malhotra, Kumar and Maheshwari [38]. These algorithms have better computational bound than the original Ford and Fulkerson's [12] algorithm.

Labelling algoring described above (Ford and Fulkerson [12]) has been coded into FORTRAN IV for the computer system IBM 7044/1401. The listing for above code can be found in Appendix B.

3.2 SHORTEST CHAIN PROBLEM:

In this section we shall discuss one of the most used algorithms in this dissertation for finding shortest chain between any pair of nodes in a network. The problem of finding shortest chains arises very frequently as a sub-problem in other problems of optimization and also has a

practical interest of its own. Every arc A_{ij} of the network has associated with it a distance c_{ij}. The problem is to find a chain from any node N_k to any other node N_1 with the sum of the distances c_{ij} of arcs in the chain a minimum.

The algorithm used by us for finding shortest chain between any pair of nodes in a network is called as 'revised cascade method' (Hu [28]). The matrix C is a N x N matrix (N is number of nodes in the network) and is called as distance matrix. At the termination of algorithm c_{ij} represents length of the shortest chain from node i to node j. C matrix provides lengths of the shortest chains at the termination where as route matrix R provides the routes for shortest chains.

The algorithm proceeds as follows. Initially all c_{ij} entries are direct distances between node i and node j.

$$c_{ii} = 0 \qquad \text{for } i = 1,\ldots, N.$$

$$c_{ij} = \infty \qquad \text{If no directed arc joining } N_i \text{ to } N_j$$
$$\text{is present.}$$

The initial R matrix has entries of $r_{ik} = k$ for each nonzero c_{ik} entry.

Now let j be fixed at $j = 1,2,\ldots, N$. For each value of j, say $j = j_0$, repeat the operation 3.2.1 for each entry c_{ik} with $i \neq j_0 \neq k$.

$$c_{ik} = \min \left\{ c_{ik} \; ; \; c_{ij} + c_{jk} \right\} \qquad (3.2.1)$$

The matrix thus generated is shortest distance matrix. To determine the shortest routes, a route matrix is generated while the shortest distance matrix is being constructed.

$$r_{ik} = \begin{cases} r_{ik} & \text{if } c_{ik} < c_{ij} + c_{jk} \\ \\ r_{ij} & \text{otherwise} \end{cases} \qquad (4.2.2)$$

This completes the description of revised cascade method for finding shortest chains. This method was coded in FORTRAN IV for IBM 7044/1401 system. The listing for the same can be found in Appendix B.

3.3 MINIMAL COST FLOW PROBLEM:

Hu [29] has given an algorithm for solving minimum cost flow problems where the shipping cost over an arc is a convex function of the number of units shipped along that arc. The problem can be started as

$$\text{Min} \quad Z = \sum c_{ij} (x_{ij}) \qquad (3.3.1)$$

$$\text{s.t.} \quad \sum_i x_{ij} - \sum_k x_{jk} = \begin{cases} -v & \text{for } j = s \\ 0 & \text{for } j \neq s,t \\ v & \text{for } j = t \end{cases} \qquad (3.3.2)$$

where $c_{ij}(x_{ij})$ are nonnegative convex functions of x_{ij}, $(c_{ij}(0) = 0)$ and the arc flows x_{ij} are required to be

positive integers or zero. The parameter v, which is required to be a nonnegative integer, represents the total flow from source to sink. Equations (3.3.2) express the conservation of flow at nodes other than source and sink.

ALGORITHM:

The algorithm for solving the minimum cost flow problem in convex cost networks can be simply described as follows. Starting with all $x_{ij} = 0$, send one unit of flow from N_s to N_t along the path whose incremental cost relative to the existing flow is minimum. This can be done by any of the existing shortest-path methods with up costs and down costs as lengths. (Up cost and Down cost for a directed arc are the cost of sending one additional unit in the direction of the arc and opposite to it respectively). In the beginning only up costs are relevant since there are no positive arc flows which could be potentially cancelled. Redefine the up costs and down costs based on the new flow pattern obtained, and send one additional unit of flow along the path with minimum incremental cost. The process of using the minimum incremental cost path is repeated until the total outflow of N_s is v (or the total inflow of N_t is v).

This algorithm in principle is used in the heuristic methods developed in Chapter IV. This algorithm is also coded in FORTRAN IV for IBM 7044/1401 system and is presented in Appendix B.

CHAPTER IV

CAPACITY EXPANSION IN SINGLE COMMODITY FLOW NETWORKS

In this chapter we will discuss various heuristic methods and an exact method developed for single commodity flow network capacity expansion problem, before that we will discuss various types of capacity expansion costs.

4.1 CAPACITY EXPANSION COSTS

Capacity of a network can be increased in two ways, either by adding some arcs to the network or by improved maintenance of arcs. This may well be explained by considering a railway network. The former case is quite obvious because addition of new links with some capacity will naturally improve capacity of the network. In the later case improved maintenance of the links in existing network will enhance trains to operate at a faster speed resulting in increased capacity of the rail network.

Capacity expansion cost function will naturally depend on the way in which capacity expansion has been achieved. Some of the cost functions which arise in practice are discussed below. Fulkerson [16] has assumed the cost of increasing capacity of a link as linear and homogeneous.

If capacity of the network is increased by adding links then total cost may look like something in Fig. 4.1.1. The sudden jumps at some interval of x_{ij} represent the cost of building a new link where as variation in cost in an interval of x_{ij} is because of maintenance and shipping costs.

If capacity of the network is increased by adding new links and if maintenance costs can almost be neglected then the total cost function may look like something in Fig. 4.1.2 or Fig. 4.1.3. In Fig. 4.1.2, shipping costs are assumed to be concave where as in Fig. 4.1.3 shipping costs are assumed to be linear.

If capacity of network is increased by adding new links and if shipping costs are negligible as compared to costs of building new links and maintenance cost then the total cost function may look like something in Fig. 4.1.3 of Fig. 4.1.4. In F ig. 4.1.3 it has been assumed that maintenance costs are linear where as Fig. 4.1.4 incorporates more common type of maintenance cost function, the convex function. If capacity of network is increased by improved maintenance of links only then total cost function will look like in Fig. 4.1.5.

In this dissertation we have assumed that maintenance and shipping costs are negligible as compared to costs of building new links. Thus total cost function which we refer in this work will look like in Fig. 4.1.6.

27

Fig. 4.1.1
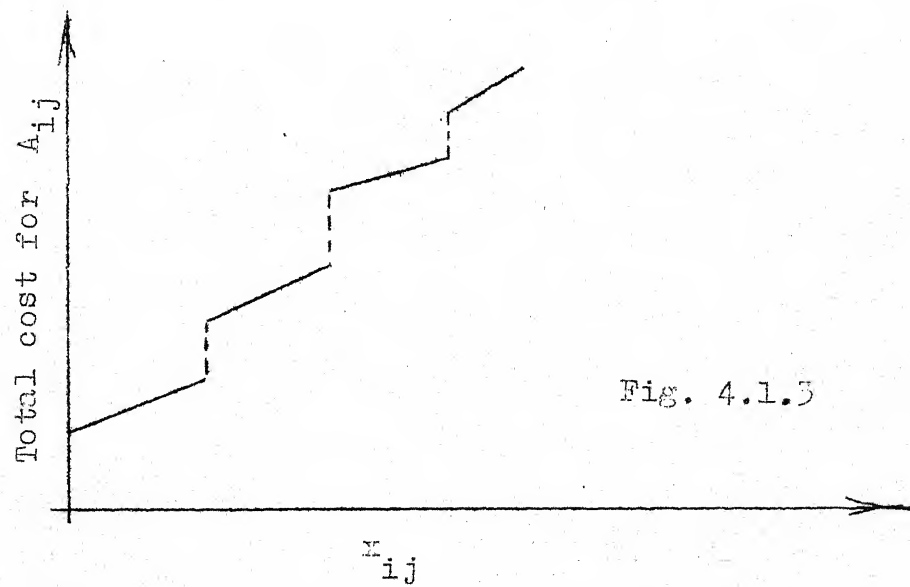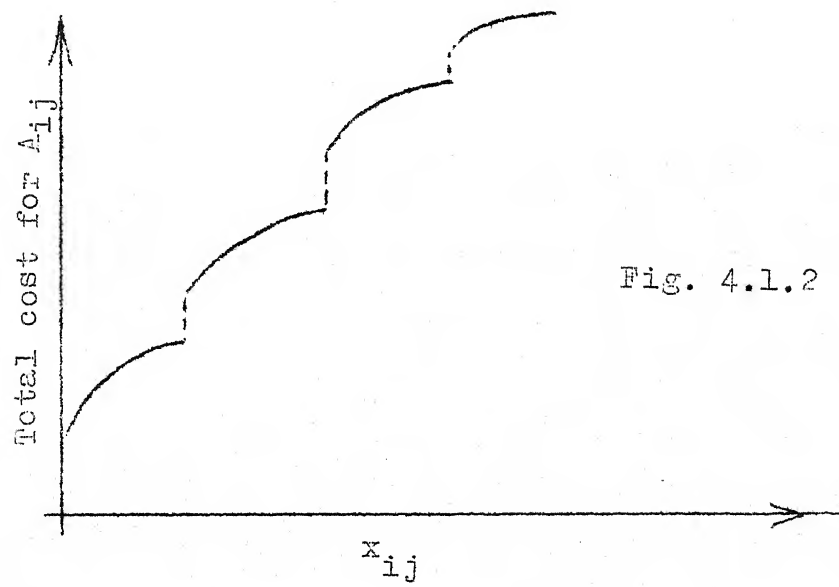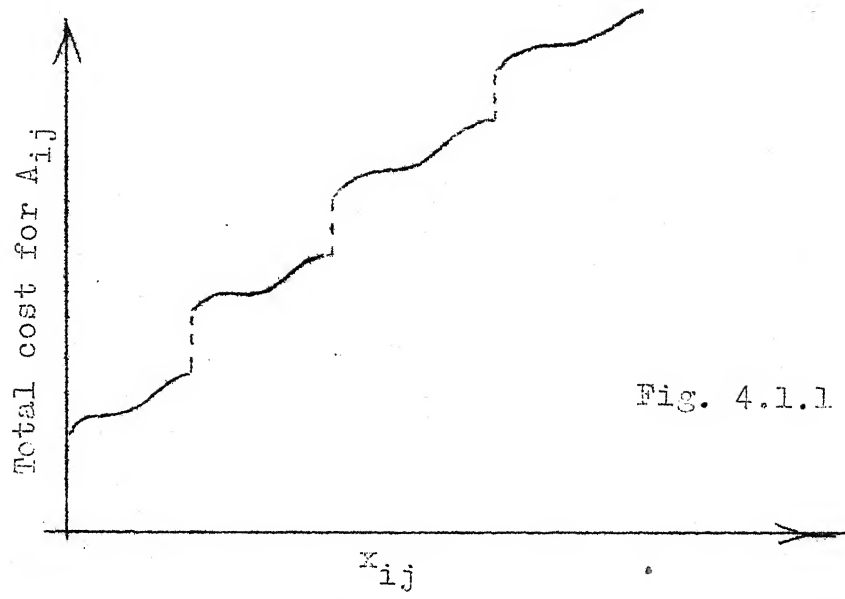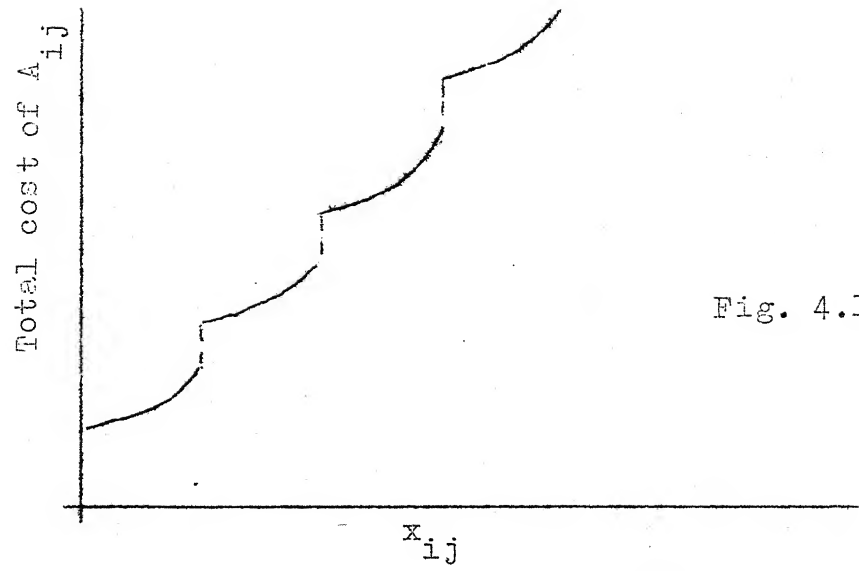
Fig. 4.1.2
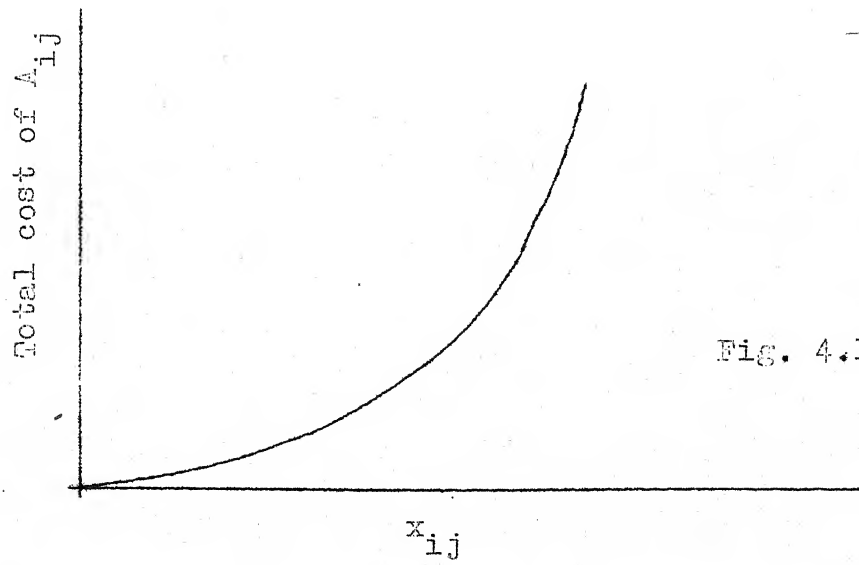
Fig. 4.1.3
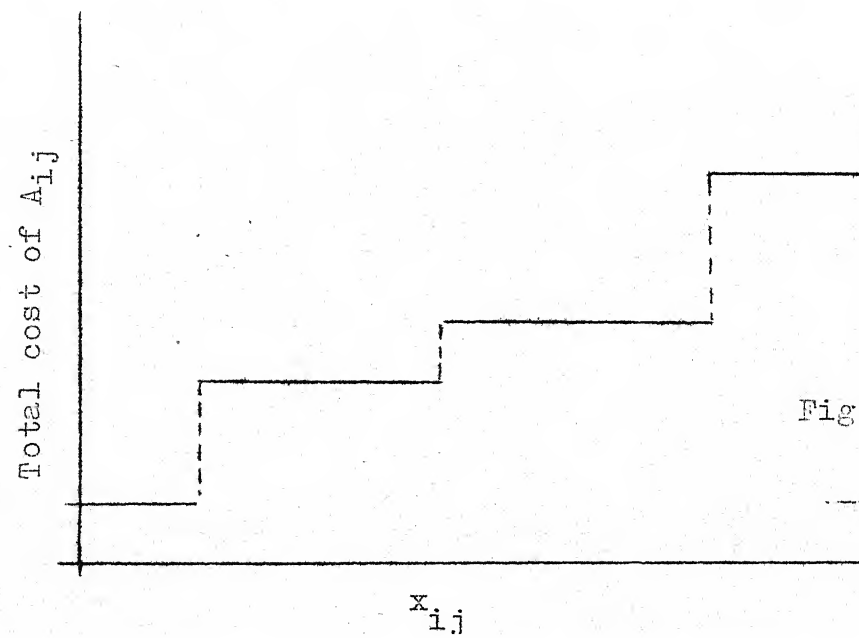
Fig. 4.1.4

Fig. 4.1.5

Fig. 4.1.6

4.2 MATHEMATICAL FORMULATION

Nomenclature:

N_i Node identification number

A_{ij} Directed arc joining N_i and N_j and heading towards N_j

b_{ij}^l Arc capacity after l-th level cost is incurred

d_{ij}^l Cost of increasing capacity from b_{ij}^{l-1} to b_{ij}^l

x_{ij} Flow on A_{ij}

V Required flow from N_s to N_t

y_{ij}^l 0-1 Variable, $y_{ij}^l = \begin{cases} 1 & \text{if } x_{ij} > b_{ij}^{l-1} \\ 0 & \text{Otherwise} \end{cases}$

can be represented mathematically as follows.

Objective Function:

The objective function is to minimize the total cost of introducing various arcs at various levels in to the network. Cost of introducing k-th level of A_{ij} can be expressed as $d_{ij}^k y_{ij}^k$. y_{ij}^k is 0-1 variable. It should be noted that A_{ij} can be introduced at k-th level only and only if A_{ij} exists in the network with k-1 levels. The total cost function, TC, can be expressed as

$$TC = \sum_i \sum_j \sum_k d_{ij}^k y_{ij}^k \qquad (4.2.1)$$

The objective function $4.2.1$ is to be optimized considering the following constraints.

Constraints:

There are two type of constraints for this problem. The first type of constraints are Kirchoff's Nodal Equilibrium constraints whereas second type of constraints arise to enforce the y_{ij}^l variables as

$$y_{ij}^l = \begin{cases} 1 & \text{if} \quad x_{ij} > b_{ij}^{l-1} \\ 0 & \text{Otherwise} \end{cases} \qquad (4.2.2)$$

Kirchoff's Nodal Equilibrium Constraints:

These constraints essentially represent the fact that flow is conserved at every node except the source and the sink. This can be represented mathematically for node j as

$$\sum_i x_{ij} - \sum_k x_{jk} = \begin{cases} -v & \text{if} \quad j = s \\ 0 & \text{if} \quad j \neq s, t \qquad (4.2.3) \\ v & \text{if} \quad j = t \end{cases}$$

These constraints will be N in number.

It should be noted that conventional capacity constraint of minimal cost flow problem is implicit in this formulation with some variation. In minimal cost flow problem arc capacity can not be increased i.e. cost associated with increasing the capacity is infinite, where as in this formulation arc capacities can be increased by another level by incurring some finite cost.

Other Constraints:

These constraints enforce y_{ij}^l variables as per 4.2.2.
These constraints can be expressed mathematically as,

$$x_{ij} - b_{ij}^{l-1} \leqslant M \, y_{ij}^l \qquad\qquad (4.2.4)$$

This type of constraints will be as many as there are y_{ij}^l
variables.

4.3 BRANCH AND BOUND METHOD (EXACT METHOD):

The problem (4.2.1) - (4.2.4) can be solved by using
any of the existing 0-1 programming algorithms.

The exact method which we are about to discuss in
this section is essentially a tree search method. For this
method there need be no specific relationship between level
costs and level capacities for all arcs.

The method starts with an upper bound set by input
feasible solution obtained by any one of the heuristic
methods of Section 4.4. If a feasible solution is not
available an arbitrarily large number is set as initial
upper bound, however a good input feasible solution may
result in sizeable saving in computational effort.

The branch and bound approach is just the simple
'divide-and-conquer' strategy. Roughly speaking it repeatedly
partitions the set of all feasible solutions in to smaller
and smaller subsets, termed 'the branching (partitioning)

process', and for each of the subsets, a lower bound for the cost of solutions within the subset is computed, termed the 'bounding process'. All the subsets are listed in a list 'L' called the 'master list'. The best known feasible solution is the 'incumbent' solution. After each parti-- tioning a subset for which, either the associated lower bound is the lowest lower bound, or it has been ascertained by some means, (e.g., if it is empty, or if the associated lower bound is greater than or equal to the objective function value for the incumbent solution), that it can not contain a feasible solution better than the incumbent solution, is discarded from all further considerations, (i.e., deleted from L) termed as the subset having been 'fathomed'. The process is continued in this fashion till either the master list 'L' is empty, or the lowest lower bound is close enough to the objective function value for the incumbent (incumbent value). On termination incumbent is an optimal solution.

In the branch and bound method discussed below branching takes place whenever lower bound on total cost is greater than or equal to the upper bound.

Whenever a feasible solution with total cost less than the present upper bound is found it sets up new upper bound on the total cost.

The following is a step wise description of the algorithm.

1.   Find a feasible solution to the problem using any of the heuristic methods discussed in next section. The total cost associated with this solution will set an upper bound U for the problem.

2.   Number the arcs (m = 1,2,..., M). Each arc can be added to network at one of t possible levels.

3.   Let all arcs be present in network, each one at its first level.

4.   Sum up all individual arc costs for present configuration to calculate total cost D. In the process of summing if at any time sum exceeds upper bound go to Step 7. Otherwise after calculating D go to Step 5.

5.   Find out maximal flow g for present configuration. Go to Step 6.

6.   If g is greater than or equal to required flow v, replace previous solution which set upper bound by present solution and go to Step 7. Otherwise go to Step 14.

7.   Set a = 1.

8.   If M - a is less than or equal to zero go to Step 9, Otherwise go to Step 10.

9.   Print present stored solution. This is optimal solution. Stop.

10.   Set h ← M−a. Increment level of h−th arc by 1. Let new level for h−th arc be f.   Go to Step 11.

11.   If f is greater than t : go to Step 12, otherwise go to Step 13.

12.   a ← a+1, go to Step 8.

13.   Set level 1 for arcs h+1 to M (both inclusive). Go to Step 4.

14.   Increment level of M−th arc by 1. Let new level for M−th arc be f.   Set h ← M. Go to Step 11.

Illustration:

Consider following problem.
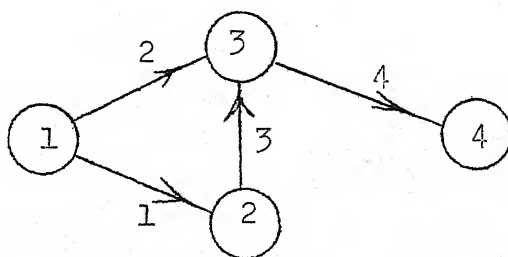
Fig. 4.1.2

Number of levels considered for each arc = t = 2.

Arc	Arc No.	1st level cost	1st level capacity	2nd level cost	2nd level capacity (Total capacity)
1, 2	1	8.0	5	3.0	10
1, 3	2	2.0	7	7.0	12
2, 3	3	5.0	3	2.0	11
3, 4	4	7.0	5	8.0	17
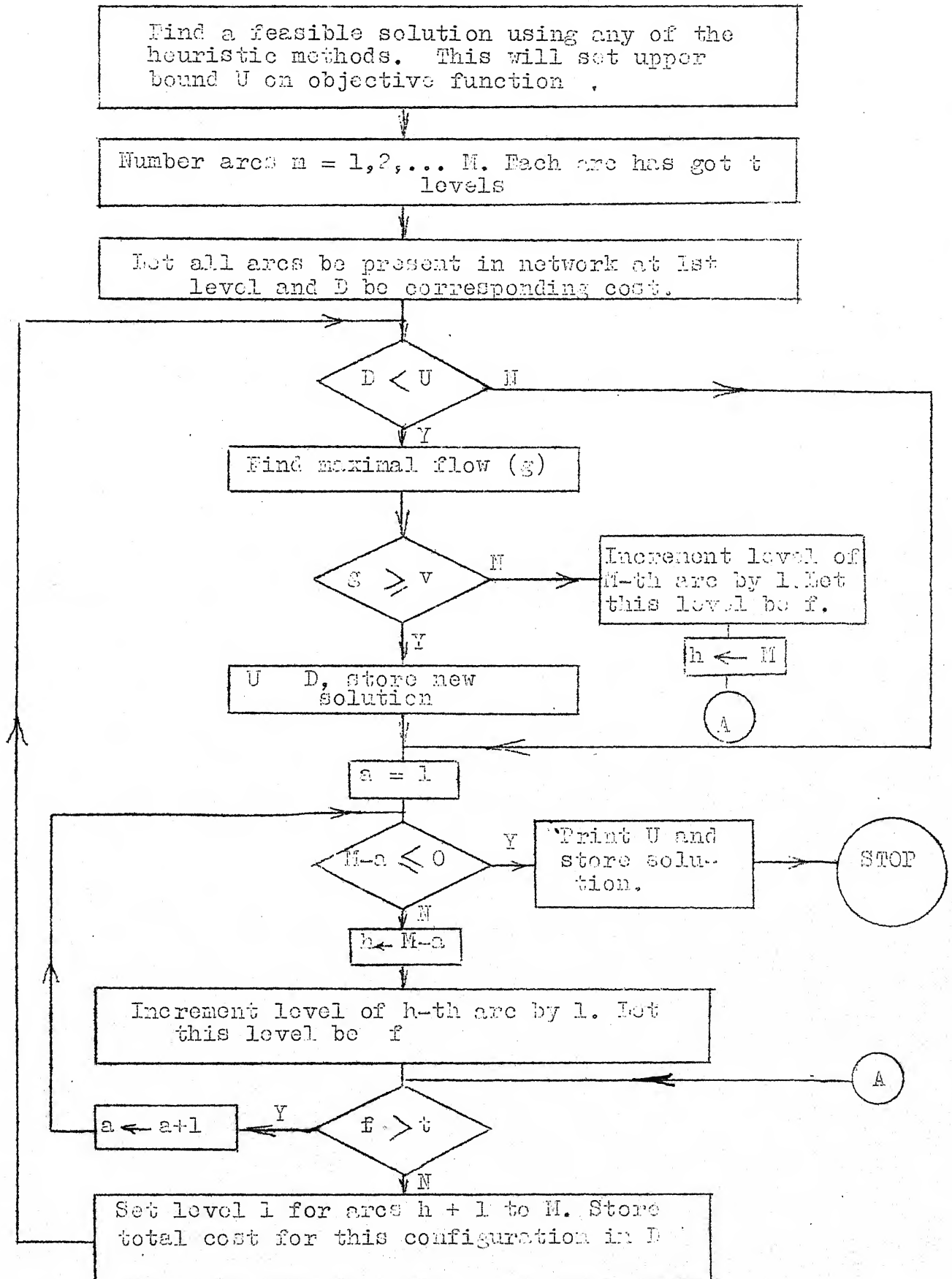
Fig. 4.3.1 Flow Digram for the Branch and Bound Algorithm

Let the required flow from node 1 to node 4 be 10.

Step 1　Since we do not have a feasible solution with us let initial upper bound on total cost be 100.0.

Step 2　Arcs are numbered as shown in the figure.

Step 3.　Let all arcs be present in the network at their first level.　Now we are at node A in the figure 4.3.3

Step 4.　Total cost D = 22. Since it is less than upper bound, we go to Step 5.

Step 5　Maximal flow g = 5.

Step 6　Since maximal flow is less than required flow this is not a feasible configuration. Hence we go to Step 14.

Step 14　Increment level of 4th arc to f = 2. Now we are at node B in the tree of figure 4.3.3. We go to Step 4.

Step 4　Total cost D for this configuration = 30.

Step 5　Maximal flow g = 10.

Step 6　Since g = v new upper bound on total cost = 30　= U.

Step 7　a = 1

Step 8　Since M - a = 4 - 1 = 3 is greater than zero we go to Step 10.

Step 10　h = 3.　Now level of 3rd arc = f = 2.

Step 11　Since f > t go to Step 13.

Step 13　Set level 1 for arc 4. Now we are at node c of tree of Figure 4.3.3.
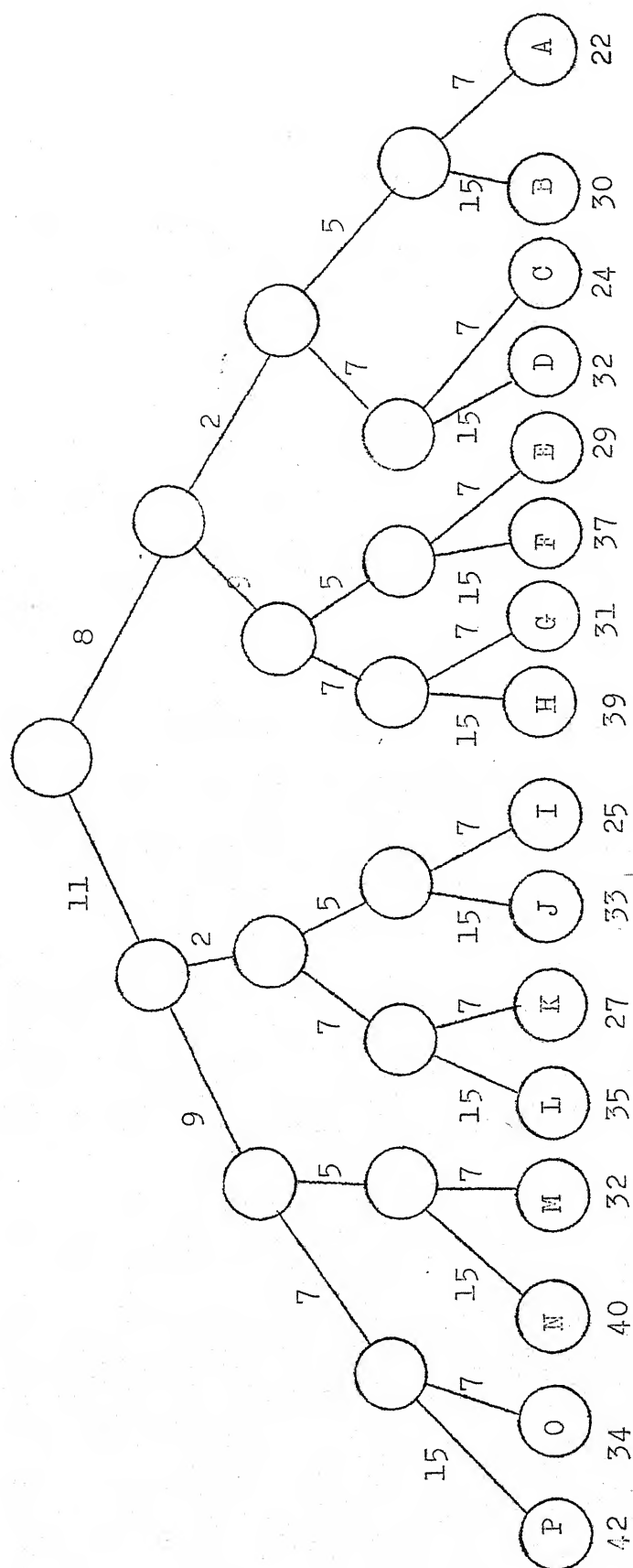
Fig. 4.3.3: Tree for the illustrative problem.

Step 4   Total cost D = 24. Since D $<$ U we go to Step 5.

Step 5   Maximal flow = g = 12.

Step 6   Since g $>$ v new upper bound on total cost = 24 = U.

We proceed in this way until all the nodes from A to P of the tree in Figure 4.3.3 are implicitly enumerated.

4.4   HEURISTIC METHODS:

Single commodity capacity expansion (SCCEP) problem can be solved by using the exact algorithm of Section 4.3 or possibly by some 0 - 1 programming algorithm. But for the exact method of Section 4.3 computations required increase at an exponential rate with the number of arcs in the network, hence it may become time consuming for networks with large number of arcs. Hence an efficient algorithm for solving the Single Commodity Capacity Expansion Problem is needed. Here we present four approximate solution techniques for solving single commodity capacity expansion problem which in the absence of an efficient exact algorithm may be proved useful.

Nomenclature:

N_i Node with index i

N_s Source node

N_t Sink node

A_{ij} Arc joining N_i and N_j and directed towards N_j

d_{ij}^{l} Cost of introducing l-th level of A_{ij} in the network.

p_{ij}^{l} capacity attained by A_{ij} after having incurred d_{ij}^{l}

d_{ij}^{k} Cost of introducing final level (k-th level) of A_{ij} into the network.

p_{ij}^{k} Capacity attained by A_{ij} after having incurred d_{ij}^{k}

b_{ij} Capacity of A_{ij}

         (b_{ij} will normally be equal to p_{ij}^{l}, $l = 1,\ldots k$)

Heuristic - 1

The algorithm first finds out single minimal cost chain from N_s to N_t which has got capacity to hold a flow greater than or equal to required flow v. It should be noted that the algorithm starts with an self imposed constraint of finding single drain which can meet flow requirement. In the later steps of algorithm arc in the chain having largest final level cost is capacitated to next lower level and cost per unit flow is updated accordingly. Other arcs in the chain are capacitated to the capacity of the existing final level and cost per unit flow is set to zero. This procedure is repeated for all arcs in the chain with their final level costs in descending order.

The above mentioned procedure for capacitating arcs and modifying c_{ij}'s makes an attempt to remove the self imposed constraint of having the solution as single chain. The following is a step wise description of the algorithm.

1. Let required flow from source to sink be v.

2. Assume that on each A_{ij} there is flow equal to v.

3. Calculate cost per unit flow c_{ij} for each A_{ij} as follows.

   c_{ij} = Cost incurred to aquire arc capacity greater than or equal to v/v.

4. Find the shortest path between N_s and the N_t with c_{ij} as distance between N_i and N_j. Let F denote the set of A_{ij} constituting the shortest path. (Let total no. of $A_{ij} \in F$ be u). This gives first feasible solution as a single chain. The total cost corresponding to this **solution sets up the upper bound U on total cost (U = D).**

5. Set q = 1,

6. Sort $A_{ij} \in F$ such that d_{ij}^k are in descending order.

7. Set w = 0.

   Calculate new c_{ij}'s and impose new b_{ij}'s for all A_{ij}'s as follows.

   a. If $A_{ij} \in F$ and ranks q-th in the ranking done above

   then, $b_{ij} = p_{ij}^{k-1}$

   $$c_{ij} = \frac{\sum_{l=1}^{k-1} d_{ij}^l}{p_{ij}^{k-1}}$$

$$d_{ij}^{k} \longleftarrow d_{ij}^{k-1}$$

$$p_{ij}^{k} \longleftarrow p_{ij}^{k-1}$$

b. If $A_{ij} \in F$ and does not rank q-th in the ranking done above,

$$b_{ij} = p_{ij}^{k}$$

$$c_{ij} = 0$$

c. If $A_{ij} \notin F$

$$b_{ij} = v$$

$$c_{ij} = \frac{\sum\limits_{l=1}^{k} d_{ij}^{l}}{v}$$

8. Using Hu's approach as described in Section 3.3, find the cost of aquiring the required flow with c_{ij} as the cost per unit flow on A_{ij} with capacity restriction b_{ij}. Also find flows on each A_{ij} this is one of the solutions. If total cost D' corresponding to this solution less than U store this new solution and set U = D'.

9. Increment q by 1 $(q \longleftarrow q+1)$

10. If q is greater than u, print out stored solution and the value of U which will be total cost value for this solution.

If $q \leqslant u$ go to Step 6.

The flow diagram for this method is as shown in

Figure 4.4.1.

Heuristic - 2:

Heuristic 1 makes an attempt at each iteration to remove one level of an arc which gives maximum decrease to the objective function. Once a level of an arc giving maximum decrease in objective function is removed no further attempt is made to improve this solution. Next iteration starts with next arc in the very first solution giving better decrease in objective function.

Heuristic - 2 differs with Heuristic 1 in the sense that it makes some attempts to improve the solution obtained by removing one of the levels of an arc in the very first solution.

The following is a stepwise description of the algorithm.

First 7 steps for this method are same as in Heuristic 1.

8.   Using Hu's approach as described in Section 3.3 find the cost of acquiring the required flow with c_{ij} as the cost per unit flow on A_{ij} with capacity restriction b_{ij}. Also find flows on each A_{ij}. This is one of the solutions. Let G be the set of A_{ij} having non zero flow on them. If D' - the total cost corresponding to this solution is less than U set U = D' and store new solution.

9. Find A_{ij}' such that $A_{ij} \in G$ and d_{ij}^k is largest.

10. Calculate new c_{ij}'s and impose new b_{ij}'s for all A_{ij}'s as follows.

a. If $A_{ij} \in G$ and equals A_{ij}'

then $\qquad b_{ij} = p_{ij}^{k-1}$

$$c_{ij} = \frac{\sum_{l=1}^{k-1} d_{ij}^l}{p_{ij}^{k-1}}$$

$$d_{ij}^k \leftarrow d_{ij}^{k-1}$$

$$p_{ij}^k \leftarrow p_{ij}^{k-1}$$

b. If $A_{ij} \in G$ and does not equal A_{ij}'

$$b_{ij} = p_{ij}^k$$

$$c_{ij} = 0$$

c. If $A_{ij} \notin G$

$$b_{ij} = v$$

$$c_{ij} = \frac{\sum_{l=1}^{k} d_{ij}^l}{v}$$

11. $w \leftarrow w+1$, If $w \geqslant 4$ go to Step 12. Otherwise go to Step 8.

12. .Increment q by 1   (q ⟵ q+1)

If q > u print the value of U, which will be the
total cost corresponding to the best solution
obtained, and the stored solution. If q ≤ u,
go to Step 7.

The flow diagram for this method is as shown in
Figure 4.4.2.

Heuristic 3:

This heuristic method is almost same as heuristic
method 1 except the criterion for sorting $A^{\#s}_{ij}$ is z_{ij}, such
that,

$$z_{ij} = \frac{d^k_{ij}}{x^k_{ij} - p^{k-1}_{ij}}$$

It was felt that capacitating an arc in the first chain
solution to its next lower level on the basis of large z_{ij}
will some times be more appropriate than merely large d^k_{ij}.
Following illustration will make this point more clear. Let
two arcs in the first chain solution have the cost structure
as in Figure 4.4.3. There is flow of 18 units of each arc.
Arc 1 has got its final level cost as 12 (23-11) and its
final level has a flow of 10 units (18 - 8). Though arc-2
has its final level cost as 10 (28 - 18), which is lower
than that of arc-1, it carries only one unit of flow.
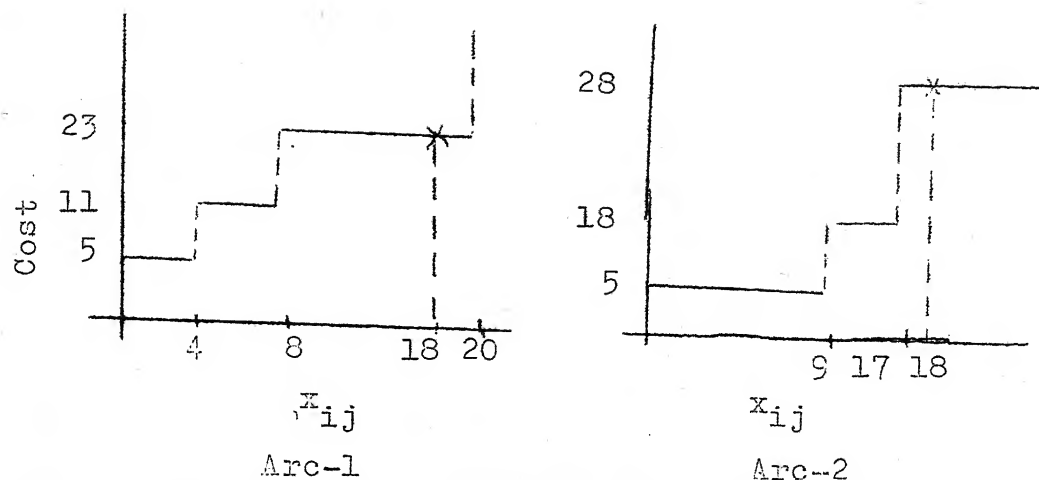
Fig. 4.4.3

It means for arc - 2 additional cost of 10 is incurred only for one unit of flow, where as arc - 1 incurs extra cost of 12 for 10 units of flow. Hence, it will seem more logical to capacitate arc - 2 at its next lower level than arc - 1 which otherwise will be capacited to its next lower level according to criterion of heuristic - 1

Heuristic - 4:

Heuristic 4 is almost same as that of heuristic 2 except the criterion for sorting $A'_{ij}s$ in step 6 is z_{ij} where z_{ij} is defined as in heuristic 3.

All these heuristic methods are coded in FORTRAN IV for IBM 7044/1401 system and are displayed in Appendix B.

4.5 Computational Performance:

The algorithms described above have been coded and tested on a number of randomly generated problems. Three problem sets were solved by each of the heuristics. The computer programs were written in FORTRAN IV for the IBM 7044/1401 system and were run using the FORTRAN compiler of that machine.

All test problems were generated as follows: starting with the required set of n nodes and an incidence matrix for the network, level costs and level capacities for each arc were generated randomly from a uniform distribution. Total number of levels generated for each arc were 4. For each node size three test problems were generated.

Table 4.5.1 shows the best solutions given by various heuristics for problem set no. 1. It can be very easily observed that heuristic-2 always gives a solution at least as good as that given by heuristic - 1. Same holds true for heuristic-4 and heuristic-3 respectively.

In many of the cases the best solutions given by heuristic-2 and heuristic-4 are not much better than that of heuristic-1 and heuristic-3 respectively. Considering the time taken by heuristic-2 or heuristic-4 to solve the large problems it seems to be a good deal to solve large problems either by heuristic-1 or heuristic-3.

But if nearness to the optimal solution even by a small amount is important heuristics 2 or 4 seem to be the desirable methodologies. Table 4.5.2 shows the average time taken to solve 3 sets of problems of various node sizes, and average number of iterations for getting the best solution given by the heuristics. Figure 4.5.1 shows the plot of average time for three sets of problems taken by heuristics 1 and 3 vs. number of nodes in the problem on linear scale. Figure 4.5.2 shows the plot of average time taken for three sets of problems by heuristics 2 and 4 vs. number of nodes in the problem on linear scale.

By observing the curves of Fig. 4.5.1 and 4.5.2 it was felt that some polynomial can be fitted in to these curves hence these data for heuristics 1 and 2 were plotted on log-log paper in Figure 4.5.3. For both the heuristics, the plots can be approximated by straight lines. These straight lines can be represented as $\log T = m \log N + \log C$ or simply $T = CN^m$ where m is the slope of the straight line in Fig. 4.5.3, C is a constant, N is number of nodes and T is computational time. For heuristic-1 computational time can be represented as,

$$T = 7.6 \times 10^{-4} \times N^{4.04}$$

For heuristic-2 computational time can be represented as

$$T = 7.49 \times 10^{-3} \times N^{3.42}$$

Because of erratic functioning of the computer system it was not possible to solve many problems with branch and bound method. However, for some of the problems solved computational time and optimal solution is shown in Table 4.5.3 along with computational time taken and best solutions obtained by various heuristics.

Percent deviation from the optimal solution for the solutions obtained by various heuristics are plotted in Fig. 4.5.4 and Fig. 4.5.5 against node size (both on linear scale).

From Fig. 4.5.4 and Fig. 4.5.5 it can be observed that heuristics - 2 and 4 provide solutions quite near to optimal solution than heuristics 1 and 3 respectively.

Fig. 4.5.6 shows the time taken by branch and bound method to give optimal solution for various node size problems.

Conclusions and Scope for Further Research:

It is quite difficult to solve the problem considered, optimally. The computational time rises steeply with problem size. Even for the linear fixed charge problem which can be thought of as our problem simplified, the available exact methods are very inefficient.

For the type of problem considered very limited work is available in the literature. Following avenues for further research are suggested.

1.   Efforts need to be made to solve the problem considered efficiently. No efficient and exact method is available for the problem.

2.   Multicommodity flow problem with cost functions considered in Fig. (4.1.1) - (4.1.5) has got resemblance with many real life problems and almost nothing has been done in this area. Thus efforts can be made to solve multicommodity flow problem with these cost functions.

Table 4.5.1

Number of Nodes	Best solutions given by			
	Heuristic-1	Heuristic-2	Heuristic-3	Heuristic-4
3	306.97	306.97	306.97	306.97
4	364.79	350.68	364.79	361.94
5	508.12	485.57	508.12	458.39
6	358.46	253.49	259.54	256.43
7	416.5	390.78	438.18	342.60
8	534.22	492.59	465.64	465.91
9	308.41	285.41	526.47	300.41
10	639.32	318.25	318.25	310.4
12	777.68	656.7	904.32	697.4
14	436.39	427.50	489.23	415.34
16	603.71	592.55	663.44	490.54
18	618.39	551.49	689.77	586.26

Table 4.5.2

Number of nodes in network	Heuristic-1		Heuristic-2		Heuristic-3		Heuristic-4	
	Time[+]	Iteration	Time[+]	Iteration	Time[+]	Iteration	Time[+]	Iteration
3	0.5	2	0.5	2	0.5	2	0.5	2
4	1.5	2.5	1	3	1	2	1.33	2.67
5	1.33	2	2	2.5	1	2.5	1.33	3
6	1	2.33	2.5	3	1	2.5	2.33	3.33
7	2.33	2.33	6.3	2.67	3	2.5	6.67	7.3
8	3.25	2.25	10	3	3.5	2	9.5	4.2
9	5	2	17	5	5	2.75	14	3
10	8	4	21.5	9	6.5	3	21.5	6.5
12	16	2	37	6	15	2	35	4
14	19	3	63.5	7	19	2	64	6
16	28	4	101	11	28	4	100	7
18	36	5	192	5	37	3	192	14

+ Average time is in IBM 7044/1401 seconds for FORTRAN compiler.

Table 4.5.3

n	p	T branch and bound	Z_o	Heuristic-1			Heuristic-2			Heuristic-3			Heuristic-4		
				T	Z	q	T	Z	q	T	Z	q	T	Z	q
3	3	2	306.97	0.5	306.97	0	.5	306.97	0	0.5	306.97	0	0.5	306.97	0
4	5	3	320.40	1	364.79	13.82	1	350.68	9.46	1	364.79	13.82	1	361.94	13
5	7	22	395.71	1	508.12	28.4	2	485.57	22.7	1	508.12	28.4	1	458.39	15.85
6	8	26	246.43	1	358.46	4.87	3	253.49	2.82	1	259.54	5.32	2	256.43	4.06
7	10	343	332.60	2	416.5	25.2	6	390.78	17.48	3	438.18	31.7	6	342.60	3

n = Number of nodes in the network

p = Number of arcs in the network

T = Computational time

Z_o = Optimal solution given by branch and bound method

Z = Best solution obtained by heuristic

q = Percentage deviation of best solution obtained by heuristic from optimal solution
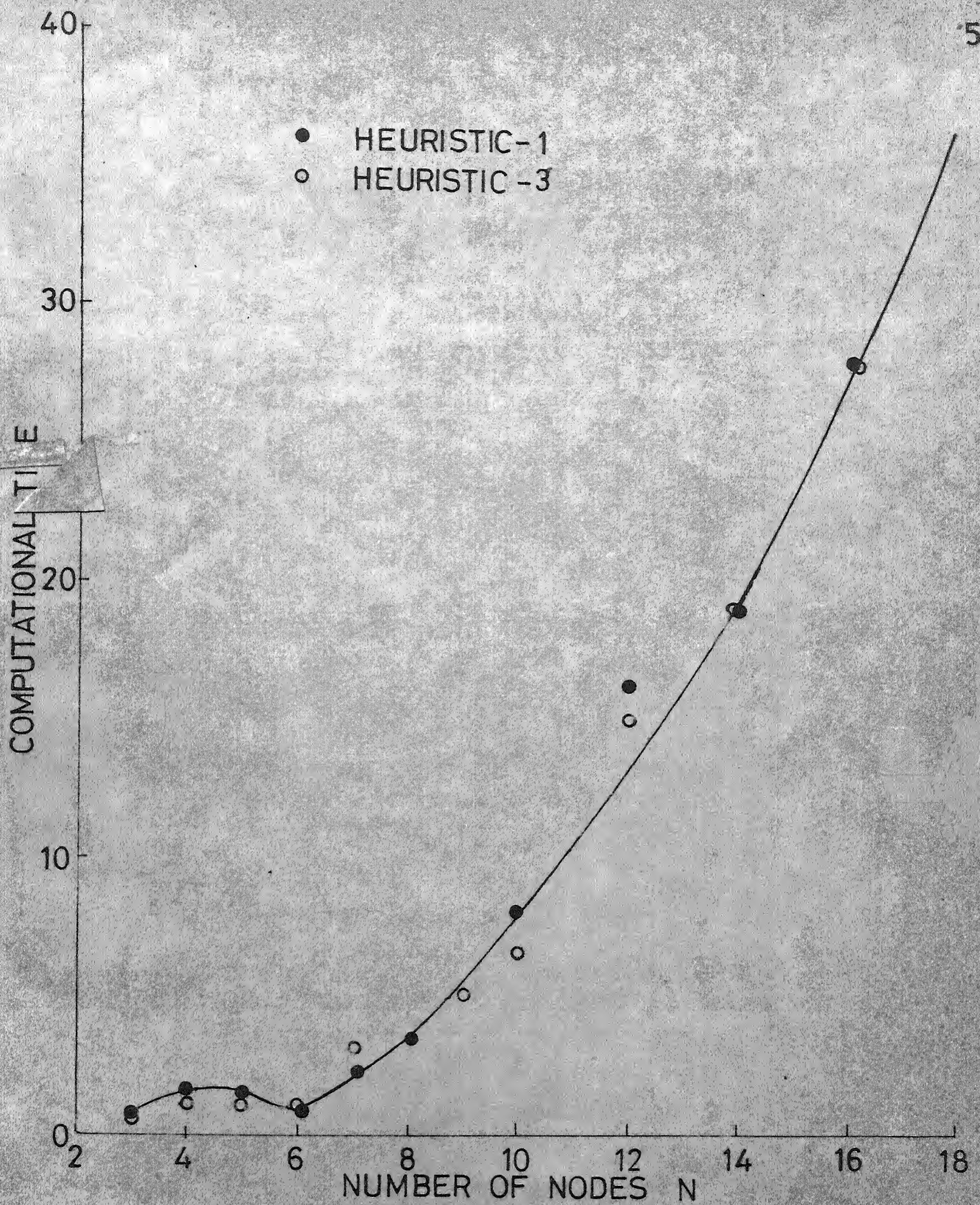
FIG. 4·5·1 PERFORMANCE OF HEURISTIC-1 AND HEURISTIC-3
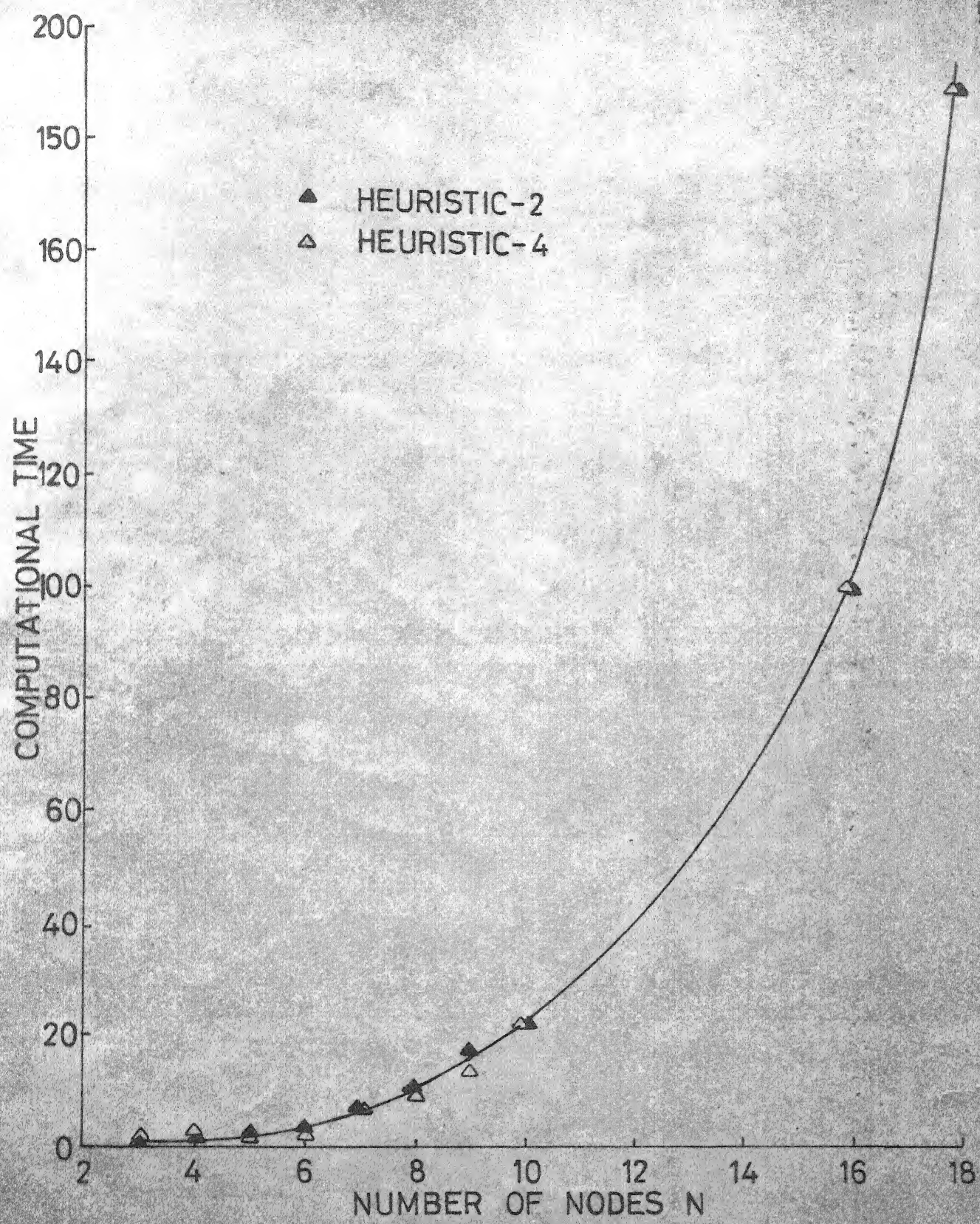
FIG. 4.5.2   PERFORMANCE OF HEURISTIC-2 AND
HEURISTIC-4
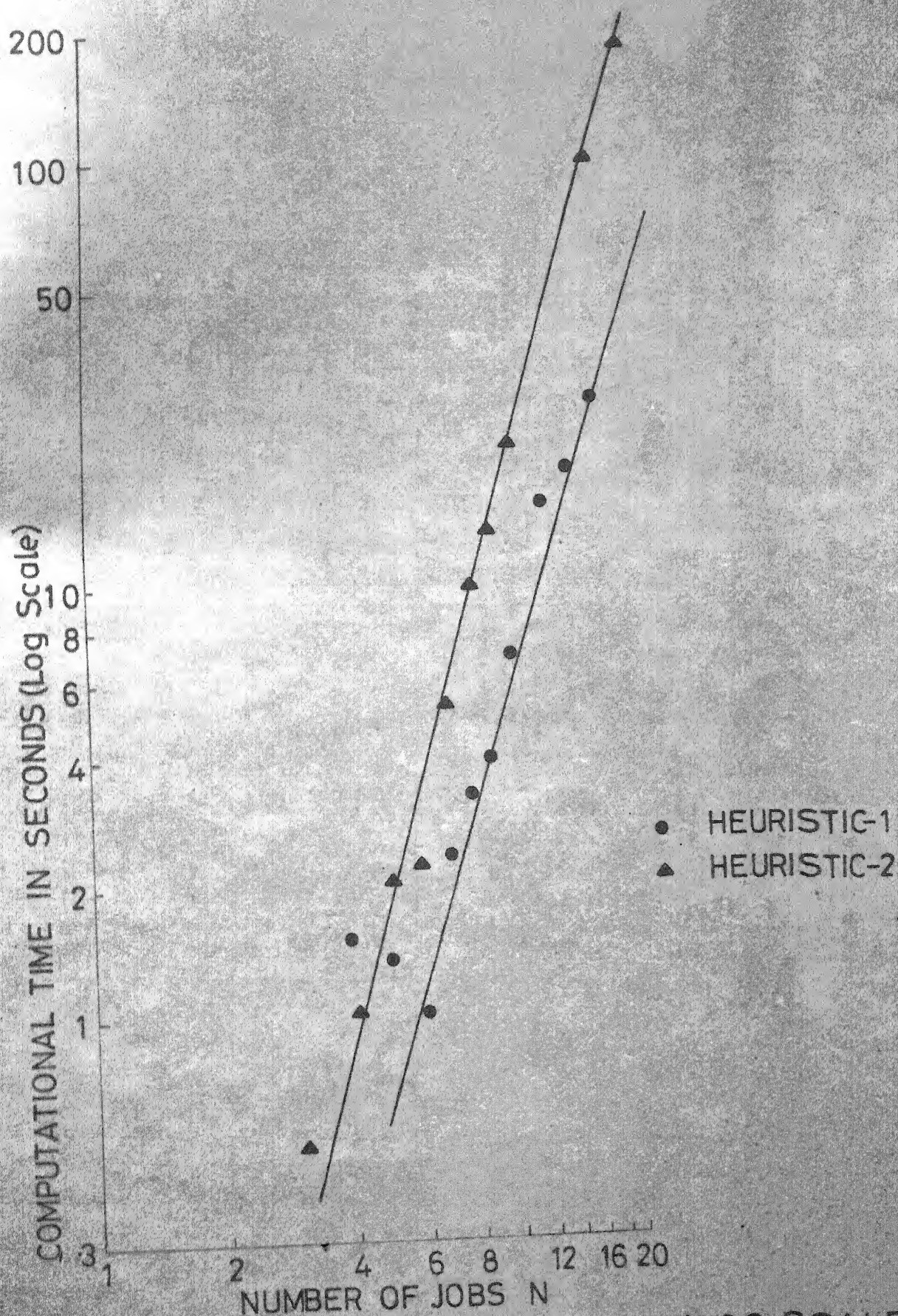
FIG. 4·5·3 COMPUTATIONAL TIME (LOG SCALE) vs NUMBER OF NODES FOR HEURISTIC-1 AND HEURISTIC-2
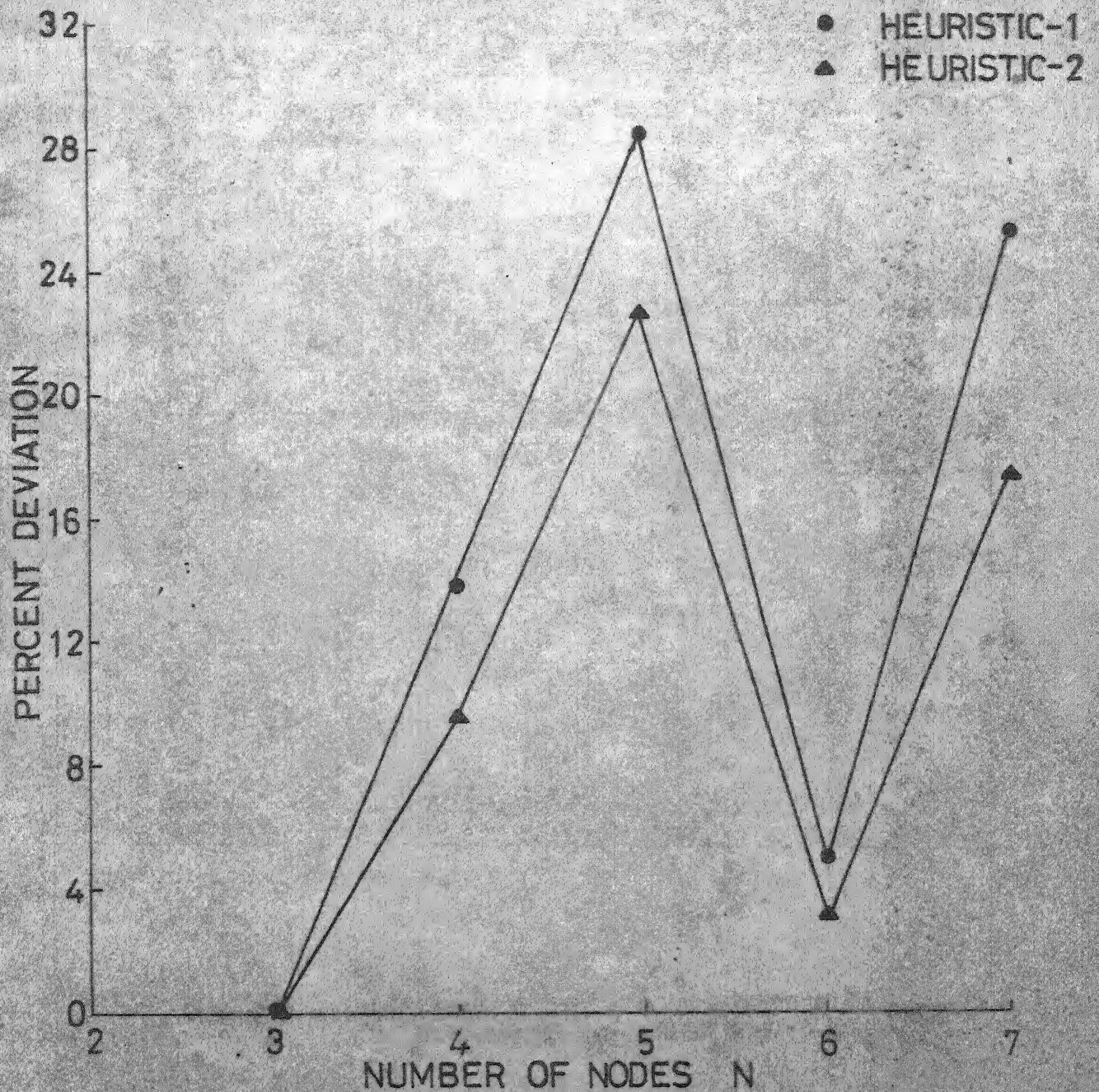
FIG. 4·5·4 PERCENT DEVIATION FROM OPTIMAL SOLUTION FOR SOLUTIONS OBTAINED BY HEURISTIC-1 AND HEURISTIC-2
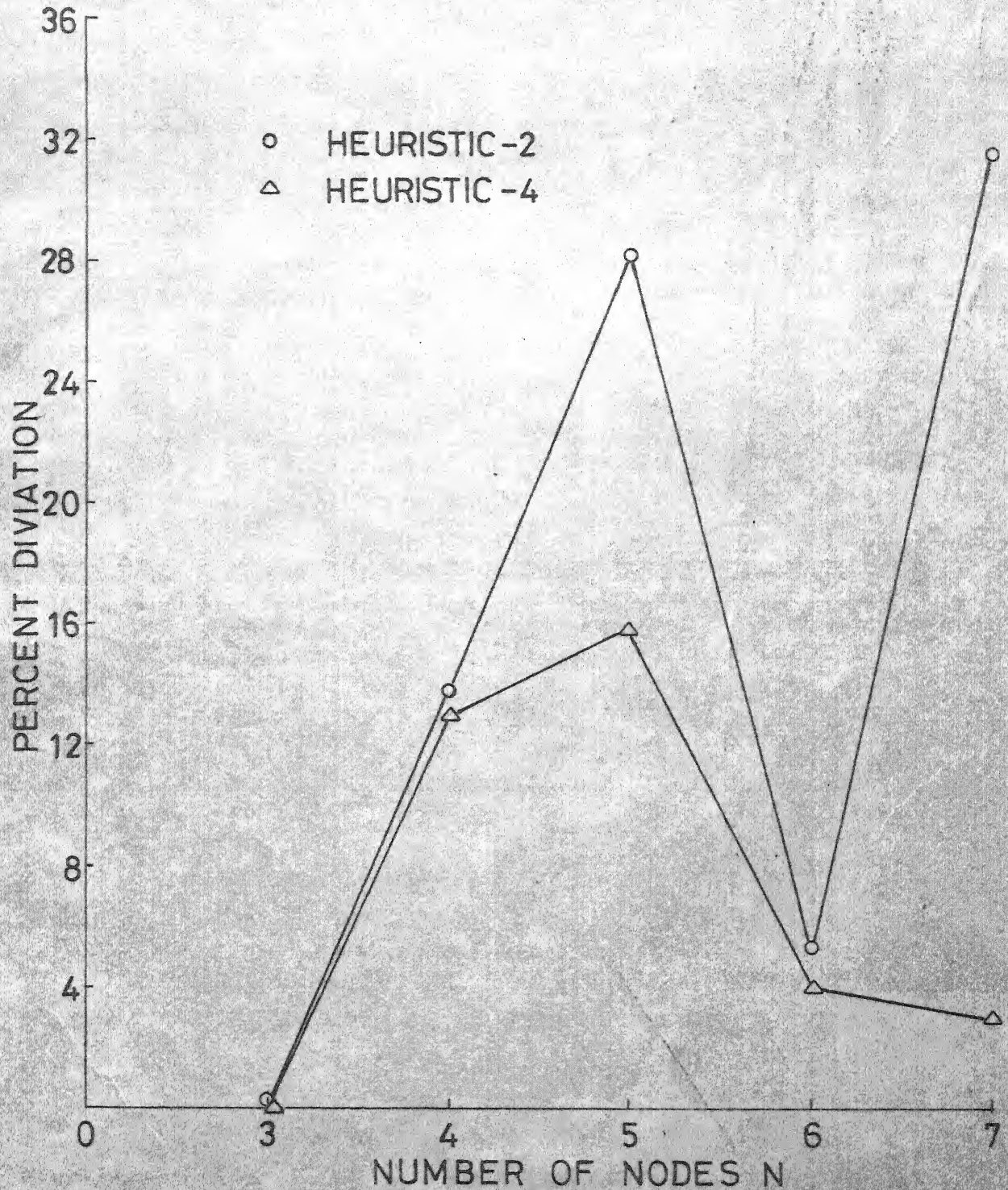
FIG. 4·5·5   PERCENT DEVIATION   FROM OPTIMAL
SOLUTION FOR SOLUTIONS OBTAINED
BY HEURISTIC-2 AND HEURISTIC-4
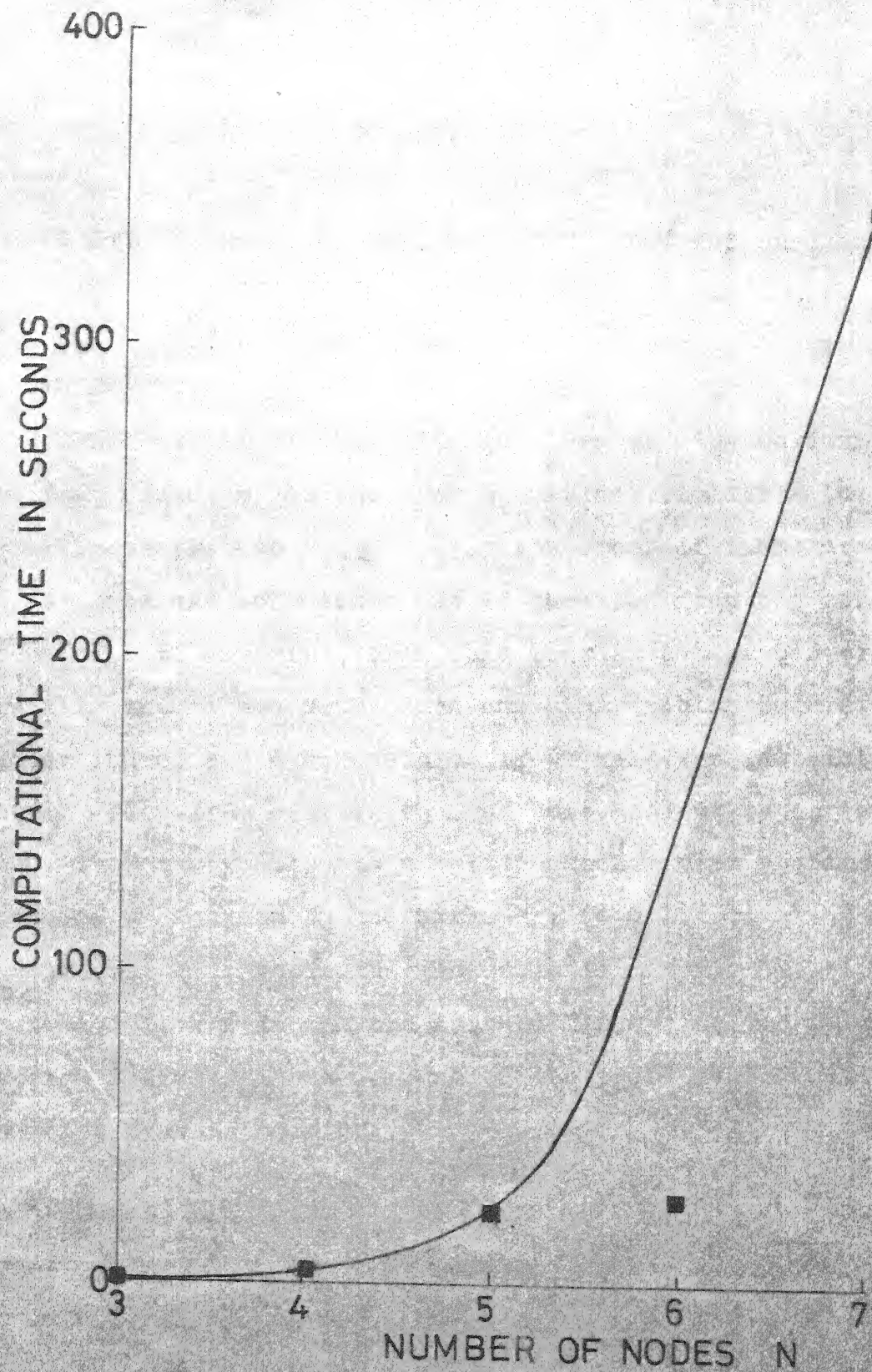
FIG. 4·5·6 PERFORMANCE OF BRANCH AND
METHOD

CAPACITY EXPANSION IN MULTICOMMODITY FLOW NETWORKS

5.1   INTRODUCTION:

The Maximal flow problem involves finding maximum flow from source N_s to the sink N_t without violating the capacity constraints $x_{ij} \leqslant b_{ij}$ on the arcs. If there are many sources and many sinks and if the flow from any source can be sent to any sink, then this problem can be converted trivially into a one source and one sink problem by creating a super source and a supersink. If we make the restriction that the flow from certain sources must be sent to certain sinks, then the problem is a multicommodity flow problem. Let there be sources N_s and sinks $N_{s'}$, $(s = 1,\ldots, q$; $s' = 1',\ldots, q')$, where the flow s is from N_s to $N_{s'}$. Let x_{ij}^s be the flow s in the arc A_{ij} and $f(s,s')$ be the value of the flow s from N_s to $N_{s'}$. One of the problems in multi-commodity flow is to find,

$$\max \quad \sum_{s=1}^{q} f(s, s') \tag{5.1.1}$$

$$\text{s.t.} \quad \sum_{i} x_{ij}^s - \sum_{k} x_{jk}^s = \begin{cases} -f(s,s') & \text{if } j = s, \\ 0 & \text{if } j \neq s, s' \\ f(s,s') & \text{if } j = s', \end{cases} \tag{5.1.2}$$

$$\sum_{s=1}^{q} |x_{ij}^s| \leq b_{ij} \qquad \forall_{ij} \qquad \qquad (5.1.3)$$

$$x_{ij}^s \geq 0 \qquad \forall_{i,j,s} \qquad \qquad (5.1.4)$$

When there is only one kind of flow, an undirected arc can always be thought of as two directed arcs of opposite direction, and arc flows of opposite directions can be cancelled. But arc flows of different commodities can not cancel each other. This is one of the central difficulties of the multi-commodity flow problem.

As said before a single commodity flow problem can always be cast into a linear program with objective function $Z = cx$ subject to the constraint $Ax = b$. The matrix A has the unimodular property, and the optimal solution has always integer components. This is no longer true in the multi-commodity flow problem. Most multicommodity flow problems cannot be solved by the labelling method or its variations.

5.2 MINIMAL COST MULTICOMMODITY FLOW PROBLEM:

The minimal cost multicommodity flow problem can be formulated in both node - arc and arc - chain form. The arc - chain formulation for minimal cost multicommodity flow problem from Tomlin [54] is presented below.

In the formulation it is assumed that there is only one source and one sink for each commodity.

Nomenclature:

a_i arc i in the network $i = 1, 2, \ldots, m$.

$c_1^k, c_2^k \ldots c_{Nk}^k$ chains of arcs joining source and sink for commodity k.

b_i Capacity of arc a_i.

c_i Cost per unit flow on arc a_i.

x_j^k flow of commodity k in chain c_j^k

a_{ij}^k $\begin{cases} 1 & \text{if } a_i \in c_j^k \\ 0 & \text{otherwise} \end{cases}$ $(j = 1, \ldots, N_k)$

$A^k = [a_{ij}^k]$ arc-chain incidence matrix.

$$i = 1, \ldots, m$$
$$j = 1, \ldots, k$$

q Total number of commodities.

r_k Requirement of commodity k.

m A very large number.

Formulation:

Objective Function:

Objective function will be minimization of total cost TC

$$\min \ TC = \sum_{k=1}^{q} \sum_{j=1}^{N_k} \sum_{i=1}^{m} c_i \, a_{ij}^k \, x_j^k \qquad (5.2.1)$$

Constraints:

Capacity constraint

$$\sum_{k=1}^{q} \sum_{j=1}^{N_k} a_{ij}^k \, x_j^k \leqslant b_i \qquad (i = 1, \ldots, m) \qquad (5.2.2)$$

Flow requirement constraint:

$$\sum_{j=1}^{N_k} x^k_j = r_k \qquad (k = 1,\ldots, q) \qquad\qquad (5.2.3)$$

The main objective of presenting above formulation is to illustrate the largeness of the linear programs for multicommodity flow problems. Normally the special structure of these programs is utilised in algorithms for their solution.[Cremeans, Smith and Tyndall [9], Saigal [48]] Hartman and Lasdon [24], Kennington and Shalaby [34].

Now before we proceed to the capacity expansion problem for multicommodity flow networks we describe the multicommodity flow feasibility problem (as described in Hu [30]).

5.3 MULTICOMMODITY FLOW FEASIBILITY PROBLEM:

In the feasibility problem of multicommodity flows, the flow value of each of the commodities is prescribed, and the question is whether or not all these flow values can be realized simultaneously in a given network. For example, let the network be as shown in Fig.(5.3.1) and the four flow requirements be as shown in Fig. (5.3.2) (i.e., we need two units of flow from N_1 to N_2, three units of flow from N_2 to N_4, etc.). We try to ship the four commodities in such a way that no arc capacity of any arc in the original network is exceeded.

Assume that we have a list of feasible networks that satisfy the flow requirements. These feasible networks all have the same number of arcs connecting the same pairs of nodes as the original given network, but every network has different arc capacities. (Some arc capacities may be zero, but we still count them as arcs of that network, so that all feasible networks have the same number of arcs.) These feasible networks are constructed as follows. For each flow requirement we find a chain from the source to the sink of that commodity. Then we assign to each arc of the chain the capacity equal to the flow requirement. Superimposing all the chains for all flow requirements, we have a feasible network.

There are millions of such feasible networks, each of which satisfies the prescribed flow requirements. Any linear convex combination of these networks also satisfies the prescribed flow requirements. The approach described below determines if the original network contains a network which is a linear convex combination of these feasible networks.

Let $[a_{ij}]$ be a matrix of m rows, each row corresponding to an arc of the given network. (In the case of Fig.(5.3.1) the matrix would be of five rows.) A column j of the matrix represents a network which has a_{1j} as the capacity of the first arc, a_{2j} as the capacity of the second arc, etc. Then all the feasible networks mentioned earlier can be represented

by columns of the matrix. If the original network represented by $[b_1, \ldots, b_m]$ contains a network $[b_1', \ldots, b_m']$ with $b_i' \leqslant b_i$ $[i = 1, \ldots, n]$, and $[b_1', \ldots, b_m']$ is a linear convex combination of the columns of $[a_{ij}]$, then the original network is feasible. Let the coefficient of the convex combination of the column j be x_j. Then the feasibility problem can be formulated as

$$\max \theta = \sum_j x_j$$

$$\text{subject to} \quad a_{ij} x_j + s_i = b_i \quad (i = 1, \ldots, m; \; j = 1, \ldots,$$

$$x_j \geqslant 0, \quad s_i \geqslant 0. \qquad (5.3.1)$$

If the optimum value of θ is 1, then the original network is feasible. There are millions of columns of columns in $[a_{ij}]$ in this formulation; fortunately, we need not write down all the columns, as the techniques like revised simplex method will prove to be of much help.
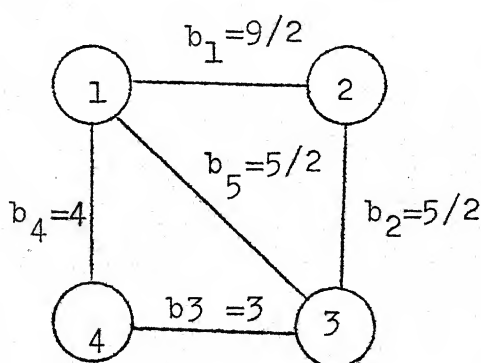
Fig. 5.3.1
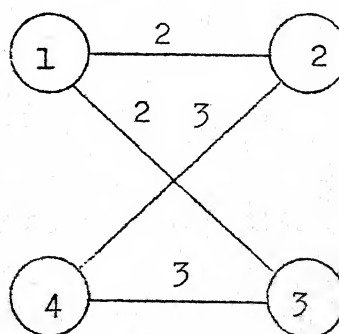
Fig. 5.3.2

5.4 MATHEMATICAL PROGRAMMING FORMULATION:

In this section we present the mathematical programming formulation for the multicommodity flow network capacity expansion problem in arc-chain form.

Objective Function:

The objective function is to minimize the total cost of introducing various arcs at various levels into the network. Cost of introducing the k-th level of a_i can be expressed as $d_{ik} \, y_{ik}$. y_{ik} is 0-1 variable. The total cost function TC, can be expressed as,

$$TC = \sum_i \sum_l d_{il} \, y_{il} \qquad (5.4.1)$$

Objective function (5.4.1) is to be optimized considering the following constraints.

Constraints:

$$\sum_{j=1}^{N_k} x_j^k = r_k \qquad (k = 1, \ldots, q) \qquad (5.4.2)$$

$$\sum_{k=1}^{q} \sum_{j=1}^{N_k} a_{ij}^k \, x_j^k - b_{il} \leqslant y_{il} \, M \qquad (5.4.3)$$

The constraints (5.4.2) are flow requirement constraints where as constraints (5.4.3) ensure that value of y_{il} is 1 when sum of all commodities flown over a_i exceeds b_{il} and value of y_{il} is 0 otherwise,

5.5 BRANCH AND BOUND METHOD:

The problem (5.4.1) - (5.4.3) can be solved by any of the existing 0-1 programming algorithms.

The branch and bound method which we have developed is exactly same as that of for single commodity flow problem, discussed in Section 4.3 except that in Step 5 we solve the multicommodity feasibility problem (5.3.1) with various arc capacities and commodity requirement vector as the input, in place of finding maximal flows.

APPENDIX A

REFERENCES

1. Balinski, M.L., 'Fixed Cost Transportation Problem', NRLQ, Vol. 8, No. 1, pp. 41-54, 1961.

2. Beale, E.M.L., 'An Algorithm for Solving the Transportation P roblems when the Shipping Cost Over each Route is Convex, NRLQ, 6, 43-46, 1959.

3. Bod, P., 'Solution of a Fixed Charge Linear Programming Problem', Proceedings of Princeton Symposium on Mathematical Programming, Princeton Univ. Press, pp. 367-375, 1970.

4. Busacker, R.G. and Gowen, P.J., 'A Procedure for Determining a Family of Minimal Cost Network Flow Problems, O.R. Office Technical Paper, 15, 1961.

5. Cabot, A.V., 'Variations on a Cutting Plane Method for Solving Concave Minimization Problem with Linear Constraints, NRLQ, Vol. 21, No.2, pp. 265-274, 1974.

6. Christofides, N. and Brooker, P., 'Optimal Expansion of an Existing Network', Mathematical Programming, Vol. 6, No. 2, 1974, 197-211.

7. Cooper, L., 'The Fixed Charge P roblem: A New Heuristic Method', Dept. of Computer Science and O.R. Southern Methodist University, Report No. CP 73026.

8. Cooper, L., and C. Drebes, 'An Approximate Solution Method for the Fixed Charge Problem', NRLQ, Vol.14, No. 1, pp. 101-113, 1967.

9. Cremeans, J.E., Smith, R.A., and Tyndall, G. R., 'Optimal Multicommodity Network Flows with Resource Allocation,' NRLQ, 17, 1970, pp. 269-280.

10. Denzler, D.R., 'An Approximate Algorithm for the Fixed Charge Problem', NRLQ, Vol. 16, No.3,pp.411-416, 1969.

11. Dinic, E.A., 'Algorithm for Solution of a Problem of Maximum Flow in a Network With Power Estimation', Sovt. Math. Dokl, Vol. 11, No. 5, 1970.

12. Ford, L.R. Jr., and Fulkerson, D.R., 'Maximal Flow Through a Network', Canadian J. Math., Vol. 8, No.3, pp. 399-404, 1956.

13. Ford, L.R. Jr. and Fulkerson, D.R., 'A Suggested Computation for Maximal Multicommodity Flows', Man. Sc. 5, 97-101 (1958).

14. Ford, L.R. Jr. and Fulkerson, D.R., Flows in Networks', Princeton University Press, Princeton, N.J. 1962.

15. Fredman, M.L.,'New Bounds on Complexity of the Shortest Path Problem', SIAM, J. Computing, Vol.5, No. 1, 1976.

16. Fulkerson, D.R., 'Increasing the Capacity of a Network: The Parametric Budget Problem', Man. Sc. 5(1959) 472-483.

17. Geoffrion, A.M. and Graves, G.W., 'Multicommodity Distribution System Design by Bender's Decomposition,' Man. Sc. Vol. 20, No. 5, 1974.

18. Golden, B.L., 'A Minimum Cost Multicommodity Network Flow Problem Concerning Imports and Exports', Networks, Vol. 5, No. 4, 1975.

19. Graves G.W. and Mcbridge R.D., 'A Factorization Approach to Large Scale Linear Programming', Working Paper No. 208, Wester Management Science Institute, University of California, Los Angeles (1973).

20. Gray, P., 'Exact Solutions of the Fixed Charge Transportation Problem', O.R. Vol. 19, No. 6, pp.1529-1538, 1971.

21. Grigoriadis, M.D. and White, W.W., 'A Partitioning Algorithm for the Multicommodity Network Flow Problems', Math. Programming, 3, 1972, pp. 157-177.

22. Hadley, G., 'Nonlinear and Dynamic Programming', Addisson-Wesley, Inc. Reading, Massachusetts.

23. Hadlock, F.O., 'A Shortest Path Algorithm for Grid Graphs', Networks, Vol. 7, No. 4, 1977.

24. Hartman, J.K. and Lasdon, L.S., 'A Generalized Upper Bounding Algorithm for Multicommodity Network F low Problems', Networks, 1, 1972, pp. 333-354.

25. Held, M., Wolfe, P. and Crowder H., 'Validation of Subgradient Optimization', Math. Programming, 6, 1974, pp. 62-68.

26. Hirsch,W.M. and Dantzig, G.B., 'Notes on L.P., Part XIX. The Fixed Charge Problem', RAND Corp., RM 1383, 1954.

27. Howard, G.T. and Nemhauser, G.L., 'Optimal Capacity Expansion', NRLQ, Vol. 15, No. 4, 1963, pp. 535-550.

28. Hu, T.C., 'Revised Matrix Algorithms for Shortest Paths in a Network', J. SIAM, Vol. 15, No. 1, pp. 207-218, 1967.

29. Hu, T.C., 'Minimal Cost Flows in Convex Cost Networks', NRLQ, Vol. 13, 1966, pp. 1-9.

30. Hu, T.C., 'Integer Programming and Network Flows', Addison-Wesley Publishing Company, Reading, Massachusetts, 1969.

31. Kaltenbach, J.C., Peschon, J. and Gehrig, E.H., A Mathematical Optimization Technique for the Expansion of Electric Power Transmission Systems', Trans. of IEEE 89 (1972) 113-119.

32. Kant, E., 'A Multicommodity Flow Problem Networks, Vol. 4, No. 3, 267-280, 1974.

33. Karzanov,A.V., 'Determining the Maximal Flow in a Network by the Method of Preflows', Sovt. Math. Dokl, Vol. 15, 1974, pp. 434-437.

34. Kennington, J., and Shalaby, M., 'An Effective Subgradient Procedure for Minimal Cost Multicommodity Flow Problems', Man. Sc. 23, 1977, 9, 994-1004.

35. Klein, M., 'A Primal Method for Minimal Cost Flows,' Man. Sc. Vol. 14, No. 3, 1967, 205-220.

36. Kleitman, D.J.,'An Algorithm for Certain Multicommodity Flow Problems', Networks, Vol. 1, No. 1, 1971,pp.75-90.

37. Le Blanc, L.J., 'Global Solutions for Nonconvex, Non-Concave Rail Network Model', Man. Sc. 23, 1976, 2, 131-139.

38. Malhotra, V.M., Kumar, M.P. and Maheshwari, S.N., 'An O(N^3) Algorithm for Finding Maximum Flows in Networks', Computer Science Programme, IIT Kanpur.

39. McKeown, P.G., 'A Vertex Ranking Procedure for Solving the Linear Fixed Charge Problem', O.R. Vol.23, No. 6, pp. 1183-1191, 1975.

40. Menon, V.V., 'The Minimal Cost Flow Problem with Convex Costs', NRLQ, Vol. 12, No. 2, 1965.

41. Murty, K.G., 'Solving the Fixed Charge Problem by Ranking the Extreme Points', O.R. Vol. 16, No.3, pp. 268-279, 1968.

42. Peart, R.M., and French, C.E., 'Optimizing Systems When Components Have Discontinuous Cost Functions', O.R. 9, 1961.

43. Pierce, A.R., 'Bibliography on Algorithms for Shortest Path, Shortest Spanning Tree, and Related Circuit Routing Problems, (1956-1974), Networks, Vol. 5, No. 2, April 1975, pp. 129-149.

44. Puri, M.C. and Kanti Swarup, 'An Algorithm for a Fixed Charge Problems', Presented at 5th Annual Convention of ORSI, Kharagpur, Dec. 1972.

45. Rardin, R.L., 'Group Theoretic and Related Approaches to Fixed Charge Problem', Dissertation in the School of Industrial and Systems Engineering, Georgia Inst. of Technology (1973).

46. Rardin, R.L. and Unger, V.E., 'Solving the Fixed Charge Network Problems, with Group Theory Based Penalties', NRLQ, Vol. 23, pp. 67-84, 1961.

47. Robacker, J.T., 'Notes on L.P.: Part XXXVII Concerning Multicommodity Networks', RM-1799, The Rand Corporation (1956).

48. Saigal, R., 'Multicommodity Flows in Directed Networks', OR Report, 66-24, ORC, Univ. of Calif. Berkeley, 1966.

49. Sakarovitch, M., 'The Multicommodity Maximal Flow Problem', ORC 66-25, O.R. Center, Univ. of Calif. Berkeley, 1966.

50. Steinberg, D.I., 'The Fixed Charge Problem',
     NRLQ, Vol. 17, No. 2, pp. 217-236, 1970.

51. Taha, H.A., 'Concave Minimization Over a Convex
     Polyhedron', NRLQ, Vol. 20, No. 3, pp. 533-548
     (1973).

52. Walker, W.E., 'A Heuristic Adjacent Extreme Point
     Algorithm for the Fixed Charge Problem', RAND Report,
     p. 5042, June 1973 or Man. Sc., Vol. 22, No. 5,
     pp. 587-596, 1976.

53. Weintraub, A., 'A Primal Approach to Network Flow
     Problems with Convex Costs', Man. Sc. 21, 1974, 1,
     Sept. pp. 87-97.

54. Tomlin J.A., Minimum Cost Multicommodity Flows,
     O.R.,Vol. 14, 1966, pp. 45.

```
$IEJCB
$IBFTC BRANCH
C        BRANCH AND BOUND
      INTEGER C(10,10)
      DIMENSION NX(10,10,10),IA(100),IB(100),IC(100),LILY(100)
      INTEGER FCLL(10,10),REQFL
      REAL JUMP(10,10,10)
      NYZ=10
      NCC=1
      READ900,DASH,NPR
  900 FORMAT(A1,I3)
  509 READ510,REQFL,N,NS,NT,NU
  510 FORMAT(20I4)
      PRINT809,REQFL,N,NS,NT,NU,NPR
  809 FORMAT(1H ,*REQUIRED FLOW =*,I4,*   NO. OF NODES=*,I4,*   SOURCE NOD
     1E=*,I4,*  SINK NODE=*,I4,* NO OF STEPS=*,I4/1H ,*PROBLEN NO.*,I4)
      NPR=2
      DO 40 I=1,N
      READ620,(FCLL(I,J),J=1,N)
      PRINT625,(FOLL(I,J),J=1,N)
  620 FORMAT(40I2)
  625 FORMAT(1H0,30I4/30I4/30I4/30I4)
   40 CONTINUE
C***  READ NETWORK COST STRUCTURE
    2 READ520,I,J,K,JUMP(I,J,K),NX(I,J,K),SIG
  520 FORMAT(3I2,F6.2,I4,A1)
C ***    PRINT NETWORK COST STRUCTURE
C ***    LAST DATA CARD HAS A DASH IN 17 TH   COLUMN
  630 FORMAT(1H0,3I2,F6.2,I4,A1)
      IF(SIG.EC.DASH)GO TO 3
      GO TO 2
    3 CONTINUE
      NK=0
      DO 20 I=1,N
      DO 20 J=1,N
      C(I,J)=0
      IF(FCLL(I,J).NE.1)GO TO 20
      NK=NK+1
      IA(NK)=I
      IB(NK)=J
      IC(NK)=1
   20 CONTINUE
      READ25,UBD
   25 FORMAT(8F10.2)
      READ620,(LILY(NIC),NIC=1,NK)
  205 SJUMP=0.0
      DO 210 I=1,NK
      LA=IA(I)
      LB=IB(I)
      LC=IC(I)
      IF(FCLL(LA,LB).NE.1)GO TO   210
      DO 207 LD=1,LC
  207 SJUMP=SJUMP+JUMP(LA,LB,LC)
      IF(SJUMP.GT.UBD)GO TO 242
  210 CONTINUE
      DO 10 I=1,NK
```

```
      LA=IA(I)
      LB=IE(I)
      LC=IC(I)
      C(LA,LB)=NX(LA,LB,LC)
   10 CCNTINUE
      PRINT625,((C(I,J),I=1,N),J=1,N)
      CALL TIME(N1)
      CALL MAXFL(N,NS,NT,C,JAST)
      CALL TIME(N2)
      IF(JAST.LT.REGFL)GC TC 240
      UBC=SJUMP
      PRINT533,UBC
  533 FCRMAT(1H ,*.... INTERMED. SCLN.COST =*,F10.3)
      PRINT625,((C(I,J),J=1,N),I=1,N)
      CALL TIME(NC)
      CC 220 I=1,MK
      LILY(I)=IC(I)
  220 CCNTINUE
      GC TC 242
  240 IC(MK)=IC(MK)+1
      IF(IC(MK).LE.NU)GC TC 205
  242 NEACK=1
  241 MKNE=MK-NBACK
      IF(MKNB)500,500,247
  247 IC(MKNB)=IC(MKNB)+1
      IF(IC(MKNB).GT.NU)GO TO 245
      MNBACK=MK-NEACK+1
  243 IC(MNBACK)=1
      MNBACK=MNBACK+1
      IF(MNBACK.GT.MK)GC TC 205
      GC TC 243
  245 NEACK=NEACK+1
      GC TC 241
  500 CC 540 I=1,MK
      LA=IA(I)
      LB=IE(I)
      LC=LILY(I)
      IF(LC.NE.0)GC TO 501
      LC=10
      NX(LA,LB,LC)=0
      GC TC 502
  501 SJUMPA=0.0
      CC 530 NN=1,LC
C-------------------
      SJUMPA=SJUMPA+JUMP(LA,LB,NN)
  530 CONTINUE
  502 PRINT630,LA,LB,LC,SJUMPA,NX(LA,LB,LC)
  540 CCNTINUE
      PRINT640,UBC
      CALL TIME(NW)
  640 FCRMAT(1H0,50(2H..)/5X,*MINIMAL COST =*,F10.3)
      CALL TIME(N1)
      IF(NCC.GE.NPR)GO TO 659
  650 NCC=NCC+1
      GC TC 509
  659 STCF
```

```
      END
$IEFTC MAXFL
      SUBROUTINE MAXFL(N,NS,NT,CD,JAST)
      INTEGER CA(20,20),B(10),A(10),C(10,10)
      INTEGER CD(10,10)
      JAST=0
      DO 1 I=1,N
      DO 1 J=1,N
    1 C(I,J)=CD(I,J)
   99 L=1
      DO 10 I=1,10
      A(I)=0
      DO 10 J=1,10
      CA(I,J)=0
   10 CONTINUE
      K=1
      I=1
      DO 100 J=1,N
      IF(C(I,J).EQ.0)GO TO 100
      A(J)=C(I,J)
      B(J)=I
      CA(L,K)=J
      IF(J.EQ.NT)GO TO 300
      K=K+1
  100 CONTINUE
  250 L=L+1
      IF(L.GE.N+2)GO TO 20
      NSU=0
      NCN=1
      K=1
  101 I=CA(L-1,NCN)
      IF(I.EQ.0)GO TO 250
      DO 200 J=1,N
      IF(C(I,J).LE.0)GO TO 158
      IF(A(J).NE.0)GO TO 200
      KAT=C(I,J)
      MAT=A(I)
      CALL MINA(KAT,MAT,MINAC)
      A(J)=MINAC
      B(J)=I
      CA(L,K)=J
      IF(J.EQ.NT)GO TO 300
      K=K+1
      GO TO 200
  158 NSU=NSU+1
      IF(NSU.EQ.N)GO TO 20
  200 CONTINUE
      NCN=NCN+1
      NSU=0
      GO TO 101
  300 LAT=B(NT)
      JAST=JAST+A(NT)
      C(LAT,NT)=C(LAT,NT)-A(NT)
      C(NT,LAT)=C(NT,LAT)+A(NT)
  305 NW=B(LAT)
      C(NW,LAT)=C(NW,LAT)-A(NT)
```

```
      C(LAT,MW)=C(LAT,MW)+A(NT)
      IF(MW.EQ.NS)GO TO 99
      LAT=MW
      GC TC 305
   20 PRINT19,JAST
   19 FORMAT(1H ,*----MAXIMAL FLOW=*,I6)
      RETURN
      END
$IBFTC MINA
      SUBROUTINE MINA(KAT,MAT,MINAC)
      IF(KAT.LT.MAT)GO TO 10
      MINAC=MAT
      GC TC 20
   10 MINAC=KAT
   20 RETURN
      END
```

```
C          MAXIMUM NUMBER OF ARCS   WHICH CAN BE CONSIDERED IS 50
C          MAXIMUM NO. OF LEVELS FOR   EACH ARC IS 4
C
C
C
C
       DIMENSION STEPA(50),IA(50),IB(50),IC(50)
       DIMENSION C(20,20),K1(20,20),CP(20,20)
       DIMENSION NX(20,20,10)
       REAL JUMP(20,20,10)
       INTEGER R(20,20),FOLL(20,20),X(20,20),CAP(20,20)
       INTEGER TP,REQFL,SUMX
       INTEGER PRN
       COST=0.0
       NYZ=20
 520   FORMAT(3I2,F6.2,I4,A1)
 600   FORMAT(20X,3I6)
       READ900,DASH,NPR
 900   FORMAT(A1,I3)
       DO 88 KKK=1,NPR
       READ510,REQFL,N,NS,NT,PRN
       PRINT809,REQFL,N,NS,NT,PRN
 809   FORMAT(1H ,3OI4)
 510   FORMAT(20I4)
       BIG=100000.0
       K=1
       DO 155 I=1,50
       STEPA(I)=0.0
 155   CONTINUE
       DO 40 I=1,N
       READ620,(FOLL(I,J),J=1,N)
       PRINT625,(FOLL(I,J),J=1,N)
 625   FORMAT(4(1H ,3OI4))
 620   FORMAT(20I2)
       DO 40 J=1,N
       C(I,J)=BIG
       CP(I,J)=C(I,J)
       X(I,J)=0
       NX(I,J,K)=IFIX(BIG)
       JUMP(I,J,K)=BIG
       K1(I,J)=1
       CAP(I,J)=BIG
 40    CONTINUE
  2    READ520,I,J,K,JUMP(I,J,K),NX(I,J,K),SIG
       PRINT630,I,J,K,JUMP(I,J,K),NX(I,J,K),SIG
 630   FORMAT(1H ,3I2,F6.2,I4,A1)
       IF(SIG.EQ.DASH)GO TO 3
       GO TO 2
  3    CONTINUE
```

```
      SJUMP=0.0
       DO 30 I=1,N
      DO 20 J=1,N
        DO 10 K=1,20
      IF(REQFL.GT.NX(I,J,K))GO TO 9
   21 K1(I,J)=K
        DO 5 K2=1,K
    5 SJUMP=SJUMP+JUMP(I,J,K2)
    6 C(I,J)=SJUMP/FLOAT(REQFL)
      CP(I,J)=C(I,J)
      SJUMP=0.0
      GO TO 20
   10 CONTINUE
    9 IF(K.EQ.4)GO TO 21
   20 CONTINUE
   30 CONTINUE
      CALL SHORT(C,N,NYZ,R)
      I=1
      TP=NS
   31 NTP1=R(TP,NT)
      K3=K1(TP,NTP1)
      COST=COST+C(TP,NTP1)*FLOAT(REQFL)
      STEPA(I)=JUMP(TP,NTP1,K3)
      IA(I)=TP
      IB(I)=NTP1
      IC(I)=K3
      I=I+1
      TP=NTP1
      IF(TP.NE.NT)GO TO 31
      ISTU=I-1
      PRINT2000,COST
 2000 FORMAT(1H0,10(2H--),F10.2)
      CALL DECEND(STEPA,ISTU,IA,IB,IC)
      MOS=1
   32 DO 50 I=1,ISTU
      LA=IA(I)
      LB=IB(I)
      LC=IC(I)
      IF(I.EQ.MOS)GO TO 34
      C(LA,LB)=0.0
      CP(LA,LB)=C(LA,LB)
      CAP(LA,LB)=NX(LA,IB,LC)
      GO TO 50
   34 SUMJUM=0.0
      LC=LC-1
      DO 41 II=1,LC
   41 SUMJUM=SUMJUM+JUMP(LA,LB,II)
      C(LA,LB)=SUMJUM/FLOAT(NX(LA,LB,LC))
      CP(LA,LB)=C(LA,LB)
```

```
      CAP(LA,LB)=NX(IA,IB,IC)
      K1(LA,LB)=LC
      TC(I)=LC
   50 CONTINUE
   81 DO 82 I=1,N
      DO 82 J=1,N
   82 C(I,J)=CP(I,J)
      CALL SHORT(C,N,NY7,R)
  290 TP=NS
      NTP1=R(TP,NT)
  300 IF(FOLL(TP,NTP1).EQ.1)GO TO 305
      IF(FOLL(TP,NTP1).EQ.(-1)) GO TO 310
  305 X(TP,NTP1)=X(TP,NTP1)+1
      IF(X(TP,NTP1).NE.CAP(TP,NTP1))GO TO 307
      C(TP,NTP1)=BIG
      CP(TP,NTP1)=C(TP,NTP1)
  307 TP=NTP1
      IF(TP.EQ.NT)GO TO 400
      NTP1=R(NTP1,NT)
      GO TO 300
  310 X(NTP1,TP)=X(NTP1,TP)-1
      IF(X(NTP1,TP).NE.0)GO TO 307
      C(TP,NTP1)=BIG
      CP(TP,NTP1)=C(TP,NTP1)
      GO TO 307
  350 CONTINUE
  400 SUMX=0
      DO 410 I=1,N
      SUMX=SUMX+X(I,NT)
  410 CONTINUE
      IF(SUMX.GE.REQFL)GO TO 360
      GO TO 81
  360 COST=0.0
      MOS1=MOS+1
      PRINT411,PRN,MOS1
  411 FORMAT(1H ,*PROBLEM NO./SOLUTION NO. =*,I3,*/*,I3)
      DO 380 I=1,N
      DO 380 J=1,N
      IF(X(I,J).EQ.0)GO TO 380
      PRINT600,I,J,X(I,J)
      DO 375 K=1,10
      IF(K.EQ.1)GO TO 374
      KT=K-1
      IF(X(I,J).LE.NX(I,J,KT))GO TO 380
  374 COST=COST+JUMP(I,J,K)
  375 CONTINUE
  380 CONTINUE
  390 PRINT 610,COST
  610 FORMAT(1H0,*TOTAL COST=*,F10.2/50(2H..))
```

```
      MCS=MOS+1
      IF(MCS.GT.ISTU)GO TO 7
      CALL INI(X,N)
      GO TO 32
    7 CALL TIME(N4)
   88 CONTINUE
      STOP
      END
C
C
C
C          SUBROUTINE SHORT FINDS SHORTEST CHAINS BETWEEN ALL PAI  OF
C      NODES IN A NETWORK
C
C
      SUBROUTINE SHORT(D,L,NYZ,R)
      DIMENSION D(20,20)
      INTEGER R(20,20)
      BIG=100000.0
      DO 110 I=1,L
      DO 110 J=1,L
      IF(D(I,J).EQ.BIG)GO TO 109
      R(I,J)=J
      GO TO 110
  109 R(I,J)=NYZ
  110 CONTINUE
      DO 140 J=1,L
      DO 130 I=1,L
      IF(I.EQ.J)GO TO 130
      DO 120 K=1,L
      IF(K.EQ.J.OR.K.EQ.I) GO TO 120
      T=D(I,J)+D(J,K)
      IF(D(I,K)-T)120,120,112
  112 R(I,K)=R(I,J)
      D(I,K)=T
  120 CONTINUE
  130 CONTINUE
  140 CONTINUE
      RETURN
      END
C
C
C
C
C
C
C
C
C          MAIN PROGRAM FOR FOR HEURISTIC 1    (SECTION 4.3)
C          MAXIMUM NUMBER OF NODES WHICH CAN BE CONSIDERED IS 10
C          MAXIMUM NUMBER OF ARCS  WHICH CAN BE CONSIDERED IS 50
```

```
C
C
C       MAIN PROGRAM FOR FOR HEURISTIC 3    (SECTION 4.3)
C           MAXIMUM NUMBER OF NODES WHICH CAN BE CONSIDERED IS 10
C           MAXIMUM NUMBER OF ARCS  WHICH CAN BE CONSIDERED IS 50
C       MAXIMUM NO. OF LEVELS FOR   EACH ARC IS 4
C
C
C
C
C
        DIMENSION STEPA(50),IA(50),IB(50),IC(50)
        DIMENSION C(20,20),K1(20,20),CP(20,20)
        DIMENSION NX(20,20,10)
        REAL JUMP(20,20,10)
        INTEGER R(20,20),FOLL(20,20),X(20,20),CAP(20,20)
        INTEGER TP,REQFL,SUMX
        INTEGER PRN
          COST=0.0
        NYZ=20
  520 FORMAT(3I2,F6.2,I4,A1)
  600 FORMAT(20X,3I6)
        READ900,DASH,NPR
  900 FORMAT(A1,I3)
        DO 88 KKK=1,NPR
        READ510,REQFL,N,NS,NT,PRN
        PRINT809,REQFL,N,NS,NT,PRN
  809 FORMAT(1H ,30I4)
  510 FORMAT(20I4)
          BIG=100000.0
          K=1
        DO 155 I=1,50
        STEPA(I)=0.0
  155 CONTINUE
        DO 40 I=1,N
        READ620,(FOLL(I,J),J=1,N)
        PRINT625,(FOLL(I,J),J=1,N)
  625 FORMAT(4(1H ,30I4))
  620 FORMAT(20I2)
        DO 40 J=1,N
        C(I,J)=BIG
        CP(I,J)=C(I,J)
        X(I,J)=0
          NX(I,J,K)=IFIX(BIG)
          JUMP(I,J,K)=BIG
          K1(I,J)=1
        CAP(I,J)=BIG
   40 CONTINUE
    2 READ520,I,J,K,JUMP(I,J,K),NX(I,J,K),SIG
        PRINT630,I,J,K,JUMP(I,J,K),NX(I,J,K),SIG
  630 FORMAT(1H ,3I2,F6.2,I4,A1)
        IF(SIG.EQ.DASH)GO TO 3
```

```
      GO TO 2
   3 CONTINUE
      SJUMP=0.0
       DO 30 I=1,N
      DO 20 J=1,N
       DO 10 K=1,20
      IF(REQFL.GT.NX(I,J,K))GO TO 9
  21 K1(I,J)=K
       DO 5 K2=1,K
   5 SJUMP=SJUMP+JUMP(I,J,K2)
   6 C(I,J)=SJUMP/FLOAT(REQFL)
      CP(I,J)=C(I,J)
      SJUMP=0.0
      GO TO 26
   9 IF(K.EQ.4)GO TO 21
  10 CONTINUE
  26 IF(FOLL(I,J).EQ.1) X(I,J)=REQFL
  20 CONTINUE
  30 CONTINUE
      CALL INI(X,N)
      CALL SHORT(C,N,NYZ,R)
      I=1
      TP=NS
  31 NTP1=R(TP,NT)
      K3=K1(TP,NTP1)
      COST=COST+C(TP,NTP1)*FLOAT(REQFL)
      IF(K3-1) 44,44,46
  44 STEPA(I)=JUMP(TP,NTP1,K3)/FLOAT(X(TP,NTP1))
      GO TO 48
  46 STEPA(I)=JUMP(TP,NTP1,K3)/FLOAT(X(TP,NTP1)-NX(TP,NTP1,K3-1))
  48 IA(I)=TP
      IB(I)=NTP1
      IC(I)=K3
      I=I+1
      TP=NTP1
      IF(TP.NE.NT)GO TO 31
      ISTU=I-1
      PRINT2000,COST
2000 FORMAT(1H0,10(2H--),E10.2)
      CALL DECEND(STEPA,ISTU,IA,IB,IC)
      MOS=1
  32 DO 50 I=1,ISTU
      LA=IA(I)
      LB=IB(I)
      LC=IC(I)
      IF(I.EQ.MOS)GO TO 34
      C(LA,LB)=0.0
      CP(LA,LB)=C(LA,LB)
      CAP(LA,LB)=NX(IA,IB,IC)
      GO TO 50
```

```
 34  SUMJUM=0.0
     LC=LC-1
     DO 41 II=1,LC
 41  SUMJUM=SUMJUM+JUMP(LA,LB,II)
     C(LA,LB)=SUMJUM/FLOAT(NX(LA,LB,LC))
     CP(LA,LB)=C(LA,LB)
     CAP(LA,LB)=NX(LA,LB,LC)
     K1(LA,LB)=LC
     IC(I)=LC
 50  CONTINUE
 81  DO 82 I=1,N
     DO 82 J=1,N
 82  C(I,J)=CP(I,J)
     CALL SHORT(C,N,NY7,R)
290  TP=NS
     NTP1=R(TP,NT)
300  IF(FOLL(TP,NTP1).EQ.1)GO TO 305
     IF(FOLL(TP,NTP1).EQ.(-1)) GO TO 310
305  X(TP,NTP1)=X(TP,NTP1)+1
     IF(X(TP,NTP1).NE.CAP(TP,NTP1))GO TO 307
     C(TP,NTP1)=BIG
     CP(TP,NTP1)=C(TP,NTP1)
307  TP=NTP1
     IF(TP.EQ.NT)GO TO 400
     NTP1=R(NTP1,NT)
     GO TO 300
310  X(NTP1,TP)=X(NTP1,TP)-1
     IF(X(NTP1,TP).NE.0)GO TO 307
     C(TP,NTP1)=BIG
     CP(TP,NTP1)=C(TP,NTP1)
     GO TO 307
350  CONTINUE
400  SUMX=0
     DO 410 I=1,N
     SUMX=SUMX+X(I,NT)
410  CONTINUE
     IF(SUMX.GE.REQFI)GO TO 360
     GO TO 81
360  COST=0.0
     MOS1=MOS+1
     PRINT411,PRN,MOS1
411  FORMAT(1H ,*PROBLEM NO./SOLUTION NO. =*,I3,*/*,I3)
     DO 380 I=1,N
     DO 380 J=1,N
     IF(X(I,J).EQ.0)GO TO 380
     PRINT600,I,J,X(I,J)
     DO 375 K=1,10
     IF(K.EQ.1)GO TO 374
     KT=K-1
     IF(X(I,J).LE.NX(I,J,KT))GO TO 380
```

```
374 COST=COST+JUMP(I,J,K)
375 CONTINUE
380 CONTINUE
390 PRINT 610,COST
610 FORMAT(1H0,*TOTAL COST=*,F10.2/50(2H..))
    MOS=MOS+1
    IF(MOS.GT.ISTU)GO TO 7
    CALL INT(X,N)
    GO TO 32
  7 CALL TIME(N4)
 88 CONTINUE
    STOP
    END
```

C
C
C
C SUBROUTINE SHORT FINDS SHORTEST CHAINS BETWEEN ALL PAIRS OF
C   NODES IN A NETWORK
C
C

```
    SUBROUTINE SHORT(D,L,NYZ,R)
    DIMENSION D(20,20)
    INTEGER R(20,20)
    BIG=100000.0
    DO 110 I=1,L
    DO 110 J=1,L
    IF(D(I,J).EO.BIG)GO TO 109
    R(I,J)=J
    GO TO 110
109 R(I,J)=NYZ
110 CONTINUE
    DO 140 J=1,L
    DO 130 I=1,L
    IF(I.EO.J)GO TO 130
    DO 120 K=1,L
    IF(K.EO.J.OR.K.EO.I) GO TO 120
    T=D(I,J)+D(J,K)
    IF(D(I,K)-T)120,120,112
112 R(I,K)=R(I,J)
    D(I,K)=T
120 CONTINUE
130 CONTINUE
140 CONTINUE
    RETURN
    END
```

C
C
C
C
C
C

```
C        SUBROUTINE INT INITIATES FLOWS ON ALL ARCS AS ZEROES
         SUBROUTINE INT(NY,N)
         DIMENSION NY(20,20)
         DO 10 I=1,N
         DO 10 J=1,N
   10    NY(I,J)=0
         RETURN
         END
C
C
C
C        SUBROUTINE DECEND SORTS DATA IN DECENDING ORDER
C
C
         SUBROUTINE DECEND(MARKS,STUDS,IA,IB,IC)
         REAL MARKS(50#+
         DIMENSION IA(50),IB(50),IC(50)
         INTEGER STUDS
         LIMIT=STUDS-1
   30    LASTSW=1
         DO 20 I=1,LIMIT
         IF(MARKS(I+1).GT.MARKS(I))GO TO 10
         GO TO 20
   10    TEMP=MARKS(I)
         MARKS(I)=MARKS(I+1)
         MARKS(I+1)=TEMP
         TEMP=IA(I)
         IA(I)=IA(I+1)
         IA(I+1)=TEMP
         TEMP=IB(I)
         IB(I)=IB(I+1)
         IB(I+1)=TEMP
         TEMP=IC(I)
         IC(I)=IC(I+1)
         IC(I+1)=TEMP
         LASTSW=I
   20    CONTINUE
         IF(LASTSW.EQ.1)GO TO 40
          LIMIT=LASTSW-1
         GO TO 30
   40    CONTINUE
         RETURN
         END
C-----------------------------------------------------------------------
```